*NASA Conference Publication 3291*

# Surface Modeling, Grid Generation, and Related Issues in Computational Fluid Dynamic (CFD) Solutions

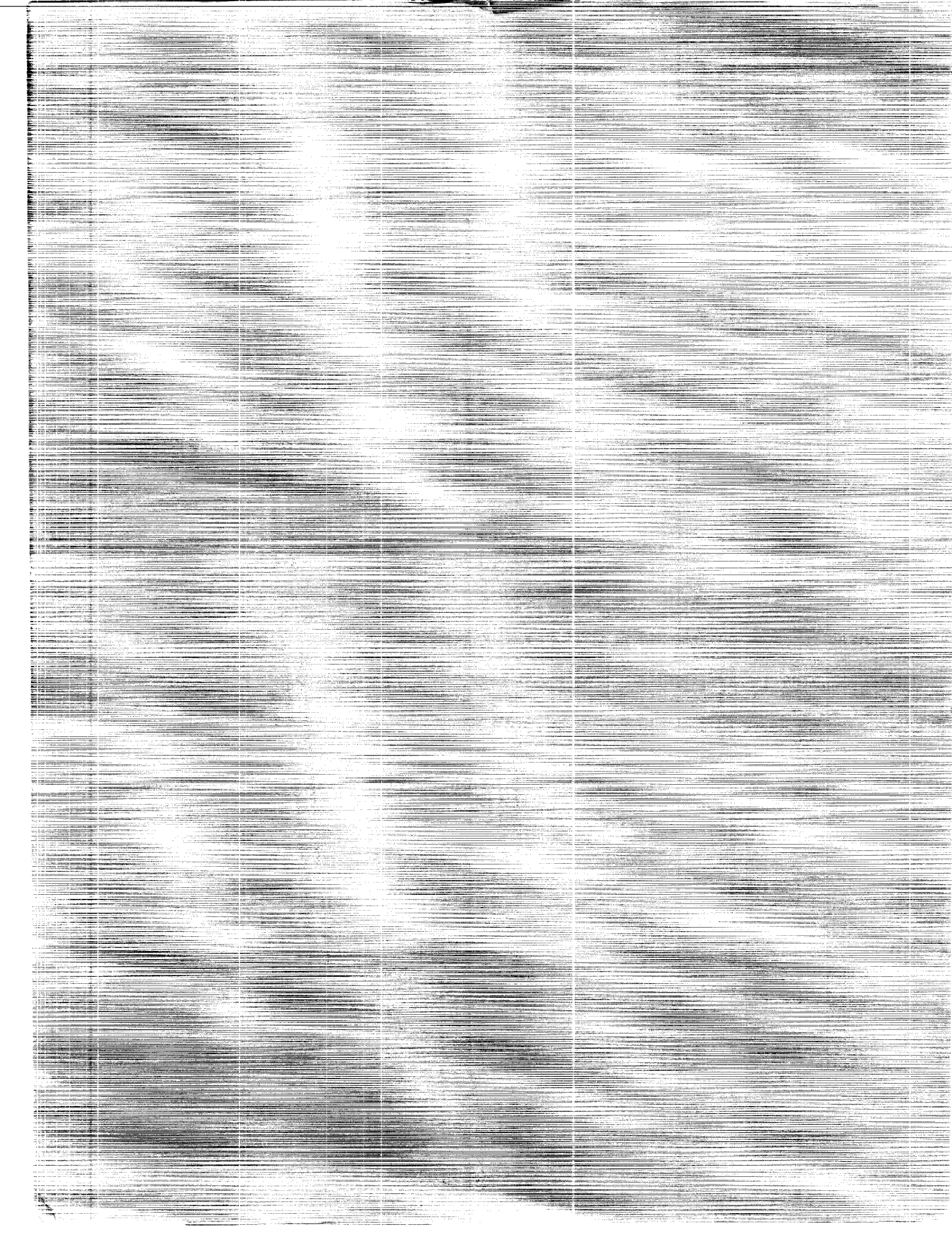
*Proceedings of a workshop held at*
*NASA Lewis Research Center*
*Cleveland, Ohio*
*May 9–11, 1995*

(NASA-CP-3291) SURFACE MODELING, GRID GENERATION, AND RELATED ISSUES IN COMPUTATIONAL FLUID DYNAMIC (CFD) SOLUTIONS (NASA. Lewis Research Center) 832 p

N95-28723
--THRU--
N95-28775
Unclas

H1/02 0048180

19951223 02



*NASA Conference Publication 3291*

# Surface Modeling, Grid Generation, and Related Issues in Computational Fluid Dynamic (CFD) Solutions

*Proceedings of a workshop*
*sponsored by the NASA Steering Committee on*
*Surface Modeling and Grid Generation*
*and held at NASA Lewis Research Center*
*Cleveland, Ohio*
*May 9–11, 1995*

National Aeronautics and
Space Administration

Office of Management

**Scientific and Technical
Information Program**

1995

# PREFACE

This publication is a collection of presentations given at the NASA Workshop on Surface Modeling, Grid Generation, and Related Issues in Computational Fluid Dynamics (CFD) solutions held at NASA Lewis Research Center, Cleveland, Ohio, May 9–11, 1995. The purpose of this workshop was to assemble engineers and scientists from industries, universities, and government laboratories and who work on geometry modeling, grid generation, computational fluid dynamics for aerospace analysis and design. Specifically, the objectives were to provide a forum on the topics; identify industry needs; assess strengths and weaknesses of technologies on structured grid (patched and overset), unstructured grid, Cartesian grid, and hybrid grid; and collectively assess for what problems each technology is uniquely suited.

This NASA Conference Publication (CP) closely follows the workshop agenda with the exception of Software Systems (1) and (2):

- Industry Overviews
- NASA Overviews
- Structured Patched Grid Technology
- Structured Overset Grid Technology
- Cartesian Grid Technology
- Unstructured Grid Technology
- Hybrid Grid Technology
- Technology Assessment
- Software Systems (1)
- Surface Grid/Geometric Grid Generation
- Automatic Zoning for Structured Multiblock Grid
- New Methods/Approaches/Applications (1)
- New Methods/Approaches/Applications (2)
- New Methods/Approaches/Applications (3)
- Software Systems (2)

A summary of panel discussions that followed each of the five grid technology sessions, software demonstrations, and small group discussions concerning future directions will be reported in a separate paper and available at a later date.


PRECEDING PAGE BLANK NOT FILMED

## WORKSHOP ORGANIZING COMMITTEE

### NASA Steering Committee on Surface Modeling and Grid Generation:

| | |
|---|---|
| Theodore G. Benjamine | NASA Marshall Space Flight Center |
| Yung Choo | NASA Lewis Research Center |
| Francis Y. Enomoto | NASA Ames Research Center |
| Austin L. Evans | NASA Lewis Research Center |
| Neal T. Frink | NASA Langley Research Center |
| Ray J. Gomez | NASA Johnson Space Center |
| Patricia A. Kerr | NASA Langley Research Center |
| Fred W. Martin | NASA Johnson Space Center |
| Cathy Maksymiuk | NASA Ames Research Center |
| Marie S. Noland | NASA Langley Research Center |
| Mark G. Potapczuk | NASA Lewis Research Center |
| Pamela F. Richardson | NASA Headquarters (Steering Committee Chair) |
| James C Townsend | NASA Langley Research Center |
| William R. Van Dalsem | NASA Ames Research Center |
| Robert W. Williams | NASA Marshall Space Flight Center |


| | |
|---|---|
| Yung K. Choo | Workshop Chair<br>e-mail: choo@lerc.nasa.gov<br>FAX: 216-433-8000<br>Phone: 216-433-5868 |
| Mark G. Potapczuk | Software Demos/Posters Session Chair<br>e-mail: tons@frosty.lerc.nasa.gov<br>FAX: 216-433-8000<br>Phone: 216-433-3919 |
| Nona Akos | Workshop Administrator<br>FAX: (216) 433-8753<br>Phone: (216) 433-8520 |

# TABLE OF CONTENTS

**INDUSTRY OVERVIEWS** ..... 1

Future Requirements in Surface Modeling and Grid Generation
Raymond R. Cosner, McDonnell Douglas Aerospace ..... 3

Requirements for Effective Use of CFD in Aerospace Design
Pradeep Raj, Lockheed Aeronautical Systems Company ..... 15

Grid Generation and Surface Modeling for CFD
Stuart D. Connell, Janet S. Sober, and Scott H. Lamson, GE Research & Development Center ..... 29

Grid Generation—A View from the Trenches
David Ives, Robert Miller, William Siddons, and Kevin Van Dyke, Pratt & Whitney ..... 45

**NASA OVERVIEWS** ..... 55

Application of CFD in Aeronautics at NASA Ames Research Center
Catherine M. Maksymiuk, Francis Y. Enomoto, and William R. Van Dalsem, NASA Ames Research Center ..... 57

GEOmetry LABoratory (GEOLAB) Surface Modeling and Grid Generation Technology and Services
Patricia A. Kerr, Robert E. Smith, and Mary-Anne K. Posenau, NASA Langley Research Center ..... 69

Surface Modeling and Grid Generation for Aeropropulsion CFD
Yung K. Choo, John W. Slater, James Loellbach, and Jinho Lee, NASA Lewis Research Center ..... 89

Marshall Space Flight Center Surface Modeling and Grid Generation Applications
Robert W. Williams, Theodore G. Benjamin, and Joni W. Cornelison, NASA Marshall Space Flight Center ..... 105

**STRUCTURED PATCHED GRID TECHNOLOGY** ..... 121

Automatic Blocking for Complex Three-Dimensional Configurations
John F. Dannenhoffer III, United Technologies Research Center ..... 123

Multiblock Grid Generation with Automatic Zoning
Peter R. Eiseman, Program Development Corporation ..... 143

Block-Structured Grids for Complex Aerodynamic Configurations: Current Status
Veer N. Vatsa, NASA Langley Research Center; Mark D. Sanetrik, AS&M, and Edward B. Parlette, ViGYAN, Inc. ..... 163

**STRUCTURED OVERSET GRID TECHNOLOGY** ..... **179**

Grid Related Issues for Static and Dynamic Geometry Problems Using Systems of Overset Structured Grids
Robert L. Meakin, Overset Methods, Inc., NASA Ames Research Center ..... 181

The Role of Overset Grids in the Development of the General Purpose CFD Code
Davy M. Belk, USAF Wright Laboratory, Eglin AFB ..... 193

**CARTESIAN GRID TECHNOLOGY** ..... **205**

A Cartesian, Cell-Based Approach for Adaptively-Refined Solutions of the Euler and Navier-Stokes Equations
William J. Coirier, NASA Lewis Research Center, and Kenneth G. Powell, University of Michigan ..... 207

Development and Application of a 3D Cartesian Grid Euler Method
John E. Melton, NASA Ames Research Center; Marsha J. Berger, Courant Institute, New York University; Michael J. Aftosmis, USAF/NASA Ames Research Center; and Michael D. Wong, Sterling Software ..... 225

Unstructured Cartesian/Prismatic Grid Generation for Complex Geometries
Steve L. Karman Jr., Lockheed Fort Worth Company ..... 251

**UNSTRUCTURED GRID TECHNOLOGY** ..... **271**

Prospects and Expectations for Unstructured Methods
Timothy J. Baker, Princeton University ..... 273

An Unstructured-Grid Software System for Solving Complex Aerodynamic Problems
Neal T. Frink, NASA Langley Research Center, and Shahyar Pirzadeh and Paresh Parikh, ViGYAN, Inc. ..... 289

**HYBRID GRID TECHNOLOGY** ..... **309**

Adaptive Hybrid Prismatic-Tetrahedral Grids for Viscous Flows
Yannis Kallinderis, Aly Khawaja, Harlan McMorris, University of Texas at Austin ..... 311

Three-Dimensional Hybrid Grid Generation Using Advancing Front Techniques
John P. Steinbrenner and Ralph W. Noack, MDA Engineering, Inc. ..... 333

**TECHNOLOGY ASSESSMENT** ..... **357**

Emerging CFD Technologies and Aerospace Vehicle Design
Michael J. Aftosmis, USAF Wright Laboratory, NASA Ames Research Center ..... 359

An Assessment of Unstructured Grid Technology for Timely CFD Analysis
Tom A. Kinard, Lockheed Aeronautical Systems Company, and Deanne M. Schabowski, Lockheed Fort Worth Company ..... 385

Evaluation of Grid Generation Technologies from an Applied Perspective
Gary S. Hufford, Vincent J. Harrand, Bhavin C. Patel, and Curtis R. Mitchell, CFD Research Corporation ..... 401

**SOFTWARE SYSTEMS (1) ..... 421**

The National Grid Project: A System Overview
Adam Gaither, Kelly Gaither, Brian Jean, Michael Remotigue, John Whitmire, Bharat Soni, and Joe Thompson, NSF Engineering Research Center, Mississippi State University; John Dannenhoffer, United Technologies Research Center; and Nigel Weatherill, University of Wales, UK ..... 423

3DGRAPE/AL: The Ames-Langley Technology Upgrade
Reese L. Sorenson, NASA Ames Research Center, and Steve J. Alter, Lockheed Engineering & Sciences Co. ..... 447

Automatic Structured Grid Generation Using GRIDGEN (Some Restrictions Apply)
John R. Chawner and John P. Steinbrenner, Pointwise, Inc. ..... 463

Tuned Grid Generation with ICEM CFD
Armin Wulf and Vedat Akdag, ICEM CFD Engineering ..... 477

**SURFACE GRID/GEOMETRIC GRID GENERATION ..... 489**

Geometry Modeling and Grid Generation Using 3D NURBS Control Volume
Tzu-Yi Yu, Bharat K. Soni, and Ming-Hsin Shih, NSF Engineering Research Center, Mississippi State University ..... 491

Generating Grids Directly on CAD Database Surfaces Using a Parametric Evaluator Approach
Timothy D. Gatzke and Thomas G. Melson, McDonnell Douglas Corporation ..... 505

Grid Generation on Trimmed Bezier and NURBS Quilted Surfaces
Chung-Jin Woan, Willard C. Clever, and Clement K. Tam, North American Aircraft Division, Rockwell International ..... 517

Geometric Grid Generation
David Ives, Pratt & Whitney ..... 535

**AUTOMATIC ZONING FOR STRUCTURED MULTIBLOCK GRID ..... 559**

Algorithms for the Automatic Generation of 2-D Structured Multi-Block Grids
Thilo Schönfeld, CERFACS, France; Per Weinerfelt, Linköping University, Sweden; and Carl B. Jenssen, SINTEF—SIMa, Norway ..... 561

Intelligent Automated Surface Grid Generation
Ke-Thia Yao and Andrew Gelsey, Rutgers University ..... 581

Optimal Domain Decomposition Strategies
Yonghyun Yoon, Korea Air Force Academy, and Bharat K. Soni,
NSF Engineering Research Center, Mississippi State University ..... 597

**NEW METHODS/APPROACHES/APPLICATIONS (1) ..... 615**

Elliptic Surface Grid Generation on Minimal and Parametrized Surfaces
S.P Spekrijse, G.H. Nijhuis, J.W. Boerstoel, National Aerospace
Laboratory NLR, Netherlands ..... 617

Cooperative Solutions Coupling a Geometry Engine and Adaptive Solver Codes
Thomas P. Dickens, The Boeing Company ..... 637

Automation of Three-Dimensional Structured Mesh Generation for
Turbomachinery Blade Passages
Edward P. Ascoli and George H.Prueger, Rocketdyne Division,
Rockwell International ..... 647

A Grid Generation System for Multi-Disciplinary Design Optimization
William T. Jones and Jamshid Samareh-Abolhassani, Computer
Sciences Corporation ..... 657

**NEW METHODS/APPROACHES/APPLICATIONS (2) ..... 669**

Automatic Multi-Block Grid Generation for High-Lift Configuration Wings
Byoungsoo Kim and Scott D. Eberhardt, University of Washington ..... 671

Multilevel Elliptic Smoothing of Large Three-Dimensional Grids
C. Wayne Mastin, NASA Langley Research Center ..... 689

Advanced Methodology for Simulation of Complex Flows Using Structured
Grid Systems
Erlendur Steinthorsson and David Modiano, ICOMP, NASA Lewis
Research Center ..... 697

**NEW METHODS/APPROACHES/APPLICATIONS (3) ..... 711**

Demonstration of An Automated CFD System for Three-Dimensional
Flow Simulations
J.W. van der Burg, J.E.J. Maseland, R. Hagmeijer, and K.M.J. de Cock,
National Aerospace Laboratory NLR, Anthony Fokkerweg 2, The
Netherlands ..... 713

A Procedure for Automating CFD Simulations of an Inlet-Bleed Problem
Wei J. Chyu, NASA Ames Research Center, and M.J. Rimlinger and
Tom I-P. Shih, Carnegie Mellon University ..... 731

A Technique for Optimizing Grid Blocks
John F. Dannenhofer III, United Technologies Research Center ..... 751

A Combined Geometric Approach for Computational Fluid Dynamics on Dynamic Grids
John W. Slater, NASA Lewis Research Center ..... 763

**SOFTWARE SYSTEMS (2) ..... 781**

Parametric Design and Gridding Through Relational Geometry
John S. Letcher, Jr. and D. Michael Shook, AeroHydro, Inc. ..... 783

Examples of Grid Generation with Implicitly Specified Surfaces Using GridPro/az3000: Filleted Multi-Tube Configurations
Zheming Cheng and Peter R. Eiseman, Program Development Corporation ..... 801

GridTool: A Surface Modeling and Grid Generation Tool
Jamshid Samareh-Abolhassani, Computer Sciences Corporation ..... 821

Interactive High Quality Grid Generation
Ning Lu and Peter R. Eiseman, Program Development Corporation ..... 833

NASA-IGES Translator and Viewer
Jin J. Chou and Michael Logan, Computer Sciences Corporation ..... 845

# INDUSTRY OVERVIEWS

19957230

N95-28724

# Future Requirements in Surface Modeling and Grid Generation

Raymond R. Cosner[1]
McDonnell Douglas Aerospace
St. Louis, Missouri

## SUMMARY

The past ten years have seen steady progress in surface modeling procedures, and wholesale changes in grid generation technology. Today, it seems fair to state that a satisfactory grid can be developed to model nearly any configuration of interest. The issues at present focus on operational concerns such as cost and quality. Continuing evolution of the engineering process is placing new demands on the technologies of surface modeling and grid generation. In the evolution toward a multidisciplinary analysis-based design environment, methods developed for Computational Fluid Dynamics are finding acceptance in many additional applications. These two trends, the normal evolution of the process and a watershed shift toward concurrent and multidisciplinary analysis, will be considered in assessing current capabilities and needed technological improvements.

## INTRODUCTION

Surface modeling and grid generation technology has long been recognized as a critical issue in practical applications of Computational Fluid Dynamics (CFD) analyses. Tools have been developed to implement these geometry modeling technologies in a reasonably versatile and efficient manner. These tools, developed for CFD applications, are rapidly gaining acceptance in additional elements of the aerospace design process: surface grid generation for processing data from pressure sensitive paint tests, surface and volume grid generation for electromagnetics and other field simulations. Technology from these thrusts also is, in a sense, returning to its roots by providing enhanced capabilities in generating surface panel networks for linear aerodynamic analyses.

In addition to the technical capabilities of the product, the development community also must consider issues of quality (i.e., fitness to intended purpose) and risk. Surface models and computational grids, of course, are not the desired end product - they are a necessary step toward producing CFD predictions or other types of data. Therefore, surface models and grids are of value only so far as they allow high-quality flow predictions to be made at an acceptable cost. Quality of the product, therefore, is determined by the CFD flow solver, and by the accuracy of the resulting flow predictions. Several types of risk must be considered in our ability to attain these products. We can identify technical risk (product may not be fit for the purpose), schedule risk (job can't be done in the planned amount of time) and budget risk (job can't be done in the allocated budget). Schedule and budget risk often derive from the use of complex tools which are inadequately tested for representative problems, or inadequately integrated into the overall design process. CFD analysis, considering the whole process, often is seen as being somewhat unpredictable in budget and schedule risk. Therefore, many program managers appreciate the benefits of CFD analysis but are unwilling to use CFD if it becomes the pacing item in the design cycle.

---

[1] McDonnell Douglas Research and Engineering Fellow. CFD Applications Manager - New Aircraft and Missile Products Division. AIAA Associate Fellow.

PRECEDING PAGE BLANK NOT FILMED

Surface modeling tools have gained great sophistication in the last ten years. However, the interface to the subsequent CFD analysis codes often is cumbersome and restrictive. Functionally, this process is unchanged since the 1980's, though it now may be carried out in a somewhat automated fashion.

Ten years ago, surface grid generation tools were highly tailored toward specific classes of geometry. High versatility was unattainable, except at a tremendous cost in calendar time and manhours. Non-interactive ("batch") computer tools were the dominant technology, and they had attained impressive power and wide acceptance. This line of technology reached its culmination in the EAGLE code, developed at Mississippi State University under USAF funding.

Batch codes such as EAGLE are capable, in the hands of an expert, of providing a suitable grid about a wide range of geometries. However, a substantial trial-and-error process often was required to integrate the surface geometry input and the batch command streams to produce a satisfactory grid. As a result, the technical capability was available, but often it could not be used on a range of high-end problems with reasonable costs, by non-specialists, producing acceptable grid quality the first time. The outcome, too often, was the frustrating situation where the expert could generate tantalizing results which could not be produced, in a practical sense, by the engineer in the design environment. Furthermore, these methods often had topological or block connectivity restrictions (e.g., point-match between blocks) which greatly reduced their usefulness in many design applications.

Part of the solution to this bottleneck was the development of interactive procedures implemented through the engineering workstation. Many efforts were initiated, through different organizations. However, the effort which exemplified this watershed technology shift was the development of GRIDGEN by General Dynamics, under USAF and NASA funding (Ref. 1). Today, in CFD technology the term "grid generation" is almost synonymous with interactive, graphics-oriented technology.

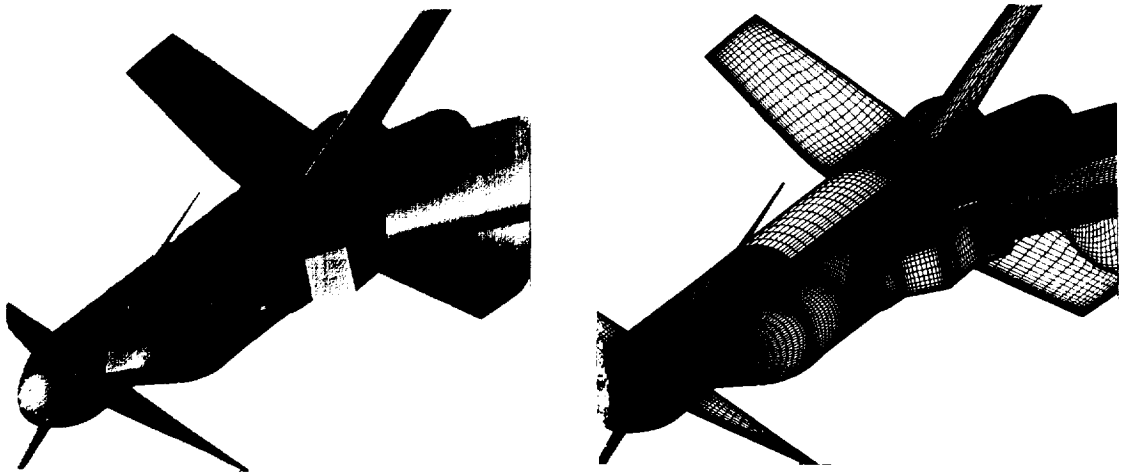


Figure 1 - Pylon/Launcher/Missile Assembly

These technologies have enabled the routine generation of usable grids about almost-arbitrary complex shapes of practical interest (as seen in Figure 1 and Figure 2). They have enabled the penetration of CFD analysis into many elements of the aerospace aero-propulsion design process.

However, the quality of the grids we can produce remains a problem (remember, quality is defined in terms of producing cost-effective flow predictions which are fit for the purpose of the study). We generally lack tools to conduct a comprehensive assessment of surface model and grid quality. The only comprehensive assessment is provided by the flow solver, which means that any defects requiring correction will inevitably have a major impact on the cost and schedule of the CFD study. Furthermore, even if we set aside quality issues, the process of surface modeling and grid generation remains a bottleneck in the total design process, consuming large resources (manhours, skilled specialists, calendar time).

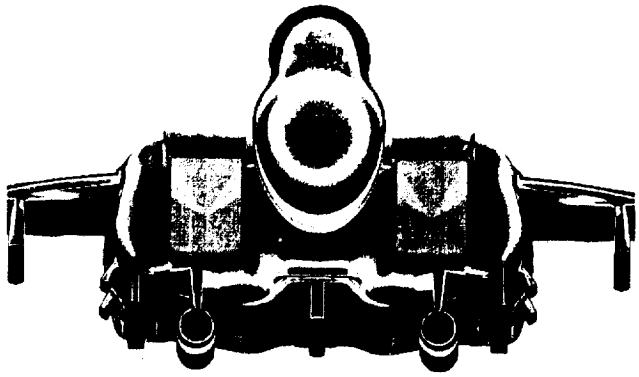


Figure 2 - F-15E Geometry Modeled for CFD Analysis (Outboard Wing not Displayed)

The observations and opinions presented here are based on the author's experience. I believe that the issues which are discussed below, in large degree, are common industry concerns. However, the material in this paper ultimately represents personal observations and opinions.

## REQUIREMENTS

**Design Process Issues** - Surface modeling and grid generation technologies, of course, do not produce a vehicle design. They are components of a complex design process. Thus, to consider future requirements for surface modeling and grid generation technology, the current limitations and future directions of the aerospace design process must be considered.

The aerospace design process is under tremendous pressure to reduce cost. Certain forms of cost, such as the value of the direct engineering labor and the capital assets used in the process, can be identified easily. Other additional costs perhaps cannot be easily evaluated, but these costs often are more critical than the direct costs to the success total design effort (see Figure 2).

***Direct Costs***
- *Engineering Manhours*
  - *Grid Generation Engineer*
  - *Specialist Support (CAD?)*
- *Value of Computer Assets*

***Additional Costs***
- *Calendar Time*
- *Highly Skilled and Specialized Labor*
- *Risks (Process Variability)*
  - *Technical Quality*
  - *Budget*
  - *Schedule*

*Identified risks - cost of mitigation plan*
*Surprises - cost of correction*

Figure 3 - Forms of Cost

A large-scale engineering project usually is very sensitive to calendar time. Economic competitiveness depends very strongly on bringing the most advanced product to the market, quickly and affordably. Further, the

ability to compress the schedule will (a) bring the product to market ahead of the competition, and (b) produce a smaller development cost. Thus, schedule compression leads to economic success in several ways.

Depending on the type of study, either the manhours or the calendar time may be the more important measure of cost. The importance of geometry acquisition and grid generation in these cost measures is illustrated in Figure 4. These data are taken from a high-end study performed in 1992-93 using Navier-Stokes structured grid methods.

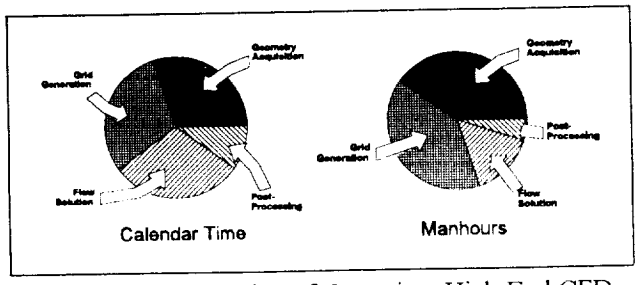



Figure 4 - Distribution of Costs in a High-End CFD Study

Generally, smaller advanced design programs are more sensitive to manhour costs, while large development programs are more sensitive to calendar time. The tasks related to geometry acquisition and grid generation consumed a substantial portion of the resources for the total task in this example. This work prior to running the flow solver code consumed about 80% of the total manhours! Clearly, in reducing the manhours and thus the direct cost of CFD analysis we should focus on the tasks of handling the geometry and building the grid. These also are important issues in reducing the calendar time of the total CFD analysis process.

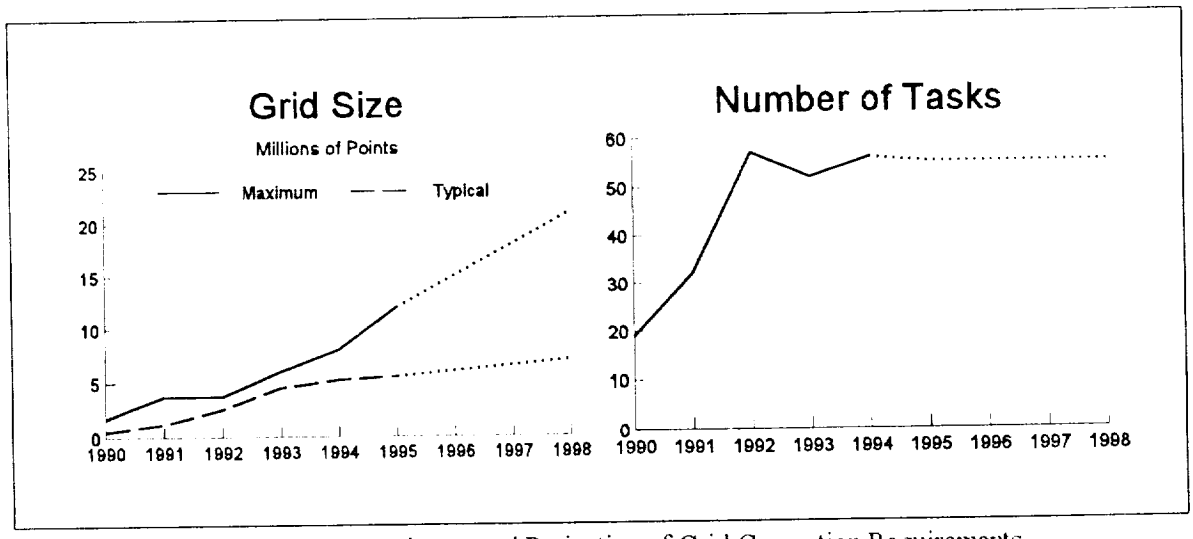



Figure 5 - Recent History and Projection of Grid Generation Requirements

Without improvements in the process, we can expect the surface modeling and grid generation phase of the process to become a far worse bottleneck in the next few years. As we succeed in establishing confidence in CFD predictions, the demands for data have increased. With these data demands have come demands for higher accuracy. One element of providing increased accuracy is to use grids with higher fidelity. This means the future grids will represent more complete modeling of the vehicle, and they will be at higher grid density to provide higher accuracy. These trends, based on our experience, are illustrated in Figure 5. Without an upturn in the overall level of engineering activity, we probably should not anticipate an increase in the number of CFD application tasks performed each year (a task is one study, consisting of a set of related grid generation and flow solution activities). However,

continuing acceptance of CFD allows the overall number of tasks to remain constant despite a generally downward trend in overall engineering activity. The demands for higher accuracy, coupled with the evolution of flow solver technology toward parallel processing platforms (allowing the solution of larger problems), provides continuing pressure toward rapid generation on ever larger and more complete computational grids.

Another "additional" cost is the need for highly skilled and specialized labor in elements of the surface modeling and grid generation process. By definition, if a certain skill is described in those terms, it is also a scarce skill. Thus, the need for specialized skills is a potential choke point in the process.

Concurrent Processes - Another strong goal is to increase the concurrency in the aerospace design process. In the jargon of computer technology, the process is shifting away from serial sequences of tasks, and shifting toward synchronized tasks spanning multiple technical disciplines. This thrust is often identified under the label of multi-discipline design or optimization. However, concurrent analysis might be a better descriptor in terms of current trends.

This thrust has several implications (additional impacts are being discovered, nearly every day). One clear implication is to identify common tools and common elements of the design process which can support multiple disciplines. Clearly, surface modeling and grid generation is a high-leverage technology in this process - it can support traditional aerodynamic analyses, wind tunnel model development, and new areas of application such as signature estimation. Surface/grid technology developed for CFD has been critical in surface mapping procedures for quantitative reduction of data from pressure sensitive paint tests.

Another, more subtle, impact of concurrent analysis is the need for high-fidelity analyses in all stages of the process. Aggressive schedules generally are not consistent with a multi-stage, hierarchical buildup in the fidelity of the supporting analyses. The new goal is to do the task once, completely and accurately, and then move to the next task. This requirement leads to a requirement for very low "latency" in the ability to produce high quality surface geometry and grids supporting the design analyses.

A third impact of concurrent engineering also must be considered. In a concurrent design process, multiple elements of the process are intertwined. A delay or failure in one element of the process will have an immediate, cascading effect throughout the process. An undetected defect in analysis products will have a much more drastic impact in a concurrent design process, compared with the impact in a more traditional sequential process. Therefore, (as always) it is important - critically important - to minimize any possibility of producing defective data. However, we must recognize that true perfection cannot be achieved. Therefore, it is also important to develop procedures to test and verify the quality of all intermediate products in the analysis process, and identify unfit products at the earliest opportunity (Figure 6).

| When Discovered | Probable Cost |
|---|---|
| Immediately | 1-2 hours, 1 person |
| During flow solution | 1 week, 1 person |
| During subsequent analysis | 1-2 months, 2-4 people |
| During design verification | 3-12 months, 4-20 people |
| During production | 1+ years, many people<br>***Product failure ???*** |

Figure 6 - Impact of a Grid Generation Defect

A schedule for elements of the design process, once established, must be maintained. The entire project (perhaps several thousand people) cannot be put in the

position of waiting for a few CFD people to complete a tardy task. But, building reserves into the schedule so that delays can be accommodated will, just as surely, produce the same non-competitive outcome - if a competitor is more successful at managing their schedule.

Risk Issues - One of the major impediments to wide acceptance of the CFD process is the perception that the process exposes the customer to high risk, i.e., that the final technical quality, schedule or budget will not be what was expected at the beginning of the CFD process. This perceived risk is mitigated by setting conservative goals (thus failing to use the full potential of the technology) and by setting aside reserves (schedule time or budget) to cover CFD variations. Thus, the current emphasis on reducing all forms of cost leads to the following goals:

- Reduce risks by eliminating process variability or uncertainty (technical quality, budget, and schedule).

- With variability under control, improve the process by reducing budget and schedule requirements, and by improving the technical quality.

## CURRENT PROCESS, ISSUES, CAPABILITY SHORTFALLS

Three broad classes of grid technology can be identified: structured grids (both overlapping and non-overlapping), unstructured grids (various types), and hybrid grids (combining structured and unstructured methods). All of these grid methods, however, obtain geometry from common sources in surface modeling.

Grid Strategy - An assessment of current issues in grid generation must consider the differing maturities of structured and unstructured grid generation. For multi-block structured grid generation, both the patched (non-overlapping) and the overset (Chimera, or overlapping) grid technologies can be said to be approaching maturity. That is, tools are available which allow these technologies to be used for (nearly) any problem by a range of engineers (i.e., not solely by experts). The standards for quality are generally understood, though achieving quality for complex problems remains difficult. Thus, the challenges in these areas, for the most part, are to improve the production capabilities of these technologies. The research goals for structured grids are process issues: elimination of bottlenecks in the work flow, improving the efficiency of the process, and ensuring that acceptable (high) quality is always achieved in the products of the process.

Unstructured grid technology is also reaching production status for several types of applications, but certainly not in the comprehensive sense in which structured technology has reached production status. With unstructured technology, process issues (see previous paragraph) are important, but other more fundamental issues of technology also can be identified. At present, Navier-Stokes unstructured grid generation for 3-D geometries remains difficult, and often requires the direct participation of the resident expert to achieve success. Standards of quality are difficult to assess and are not fully understood, except in the most clear-cut cases. At present, we usually rely on the capabilities of the grid generation tools as demonstrated in relatively simple problems. We often cannot evaluate the quality of the grid product except by the behavior and the product of the flow solver. Visual inspection of the volume grids, prior to beginning the flow solution, is virtually worthless. Useful quality standards are not accepted, and quality assurance tools are almost non-existent.

An intermediate level of technology, hybrid structured and unstructured grids, has received relatively little attention. This technology, perhaps, offers an operational compromise. It might permit the engineering community to use proven viscous flow methods on structured grids near walls (maybe restricted to very near the walls), and take advantage of the versatility of unstructured methods in complex, multiply-connected volume regions between vehicle components.

Domain Decomposition - For multi-block structured grid methods, volume grid generation must be preceded by definition of the block boundaries. In overset grid methods, the exact location of block boundaries away from the vehicle surface may not be a critical issue. Hyperbolic grid generation procedures often are used, due to their high efficiency and the grid quality which they now can produce (since the location of the outer edge of the grid block is usually not crucial). However, this overset grid approach leads to difficult issues in generating the boundary condition coupling (interface) data between communicating grid blocks (more on this later).

With structured/patched grid methods (i.e., non-overlapping or marginally overlapping grids), the locations of block boundaries are quite important. These boundaries must be defined across the computational domain at about the same point in the process where the surface grids are generated. This process of defining block boundaries, often called "domain decomposition," consumes much time (both calendar time and manhours). Several research efforts are underway with the goal of developing automated tools for domain decomposition, often with the aid of artificial intelligence technologies. As an intermediate step, perhaps the techniques of 3-D visualization can be used with interactive cutting planes to define block boundaries quickly (in seconds or minutes).

Surface Modeling - For several years, the preferred source of surface geometry has been Computer-Aided Design (CAD) systems, such as the Unigraphics system used at McDonnell Douglas. Often, the CAD geometry must be edited - to correct defects, to trim the surfaces (i.e., to eliminate non-physical edges of surface elements), or to modify the true geometry for the purposes of the analysis. Next, the geometry often is converted to closely-spaced mathematical section cuts or a pointwise definition for use in the CFD grid generation system.

Usually, the geometry is first defined within the CAD environment as a wireframe model. Next, the wireframe is surfaced (i.e., all geometry is defined to produce a complete 3-D definition of the exposed surfaces). This step of CAD surfacing can be time consuming, particularly if higher-order constraints must be enforced for continuity in surface slope or curvature across abutting surface elements.

CAD geometry ideally consists of surface patches or volume elements which abut cleanly, with no gaps, overlaps, doubly defined regions, or non-physical protrusions. In reality, these and other types of defects occur, as are illustrated in Figure 7.

Correcting these geometry definition defects is perhaps the chief bottleneck of the process. This is a non-value-added step. Ideally, surface modeling tools would use safeguards to avoid generating these defects in the geometry.

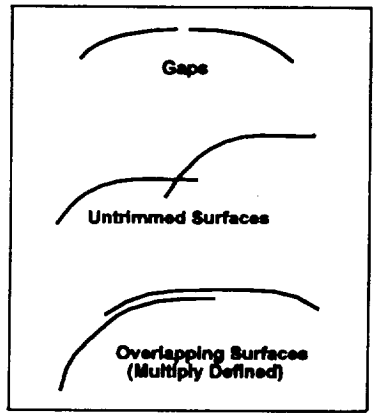



Figure 7 - Examples of Surface Geometry Defects

One needed step is to develop surface modeling tools, outside the CAD environment, which can either correct these defects or generate a suitable surface grid despite the defects. However, a fundamental issue remains: which definition is to be used where the surface is multiply defined? Can an automated algorithm be established which can determine whether a protrusion in the surface geometry is correct or a defect? Can an automated algorithm fully address the issue of gaps in the geometry - what if the gaps are intentional (for example, inlet bleed slots)? It seems that semi-automated tools are needed, to locate potential defects for human inspection, with automated correction depending on the outcome of the inspection. The development of tools which are tolerant of surface defects would greatly improve the cycle time of the CFD analysis process. Some of these issues are summarized in Figure 8.

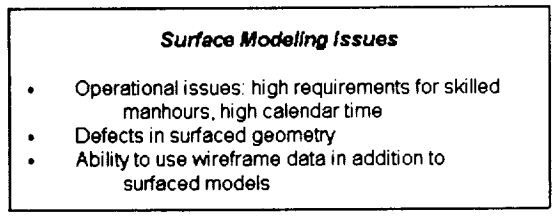



Figure 8 - Surface Modeling Issues

Another needed step is to develop tools which allow grid generation to begin with an arbitrary wireframe model, rather than a fully surfaced model. Perhaps high precision would not be required in slope continuity across surface patch abutments, and for CFD purposes it probably would not be necessary to provide continuity of curvature. These tools would be useful mainly in the advanced design environment; for more accurate data in the later stages of design it would be necessary to use "official" CAD surface geometry for consistency. The ability to use arbitrary wireframe data as input to the grid generator, for advanced design purposes, would greatly improve CFD turnaround.

A third approach, which is gaining popularity, is to base the grid generation process on a 3-D surfaced model external to the CAD system. The NASA IGES format (Ref. 2) is gaining favor in this role.

Surface Grid Generation - The next step in the process is surface grid generation. This step, for the most part, is common for all grid methods. A quality surface grid, or mapping, must be applied to all geometry surfaces which are to be retained in the analysis. For most methods, this step requires careful definition of all intersection lines between components. This is the step which becomes highly subjective if defects remain in the surface modeling. Some issues and requirements for this step are presented in Figure 9.

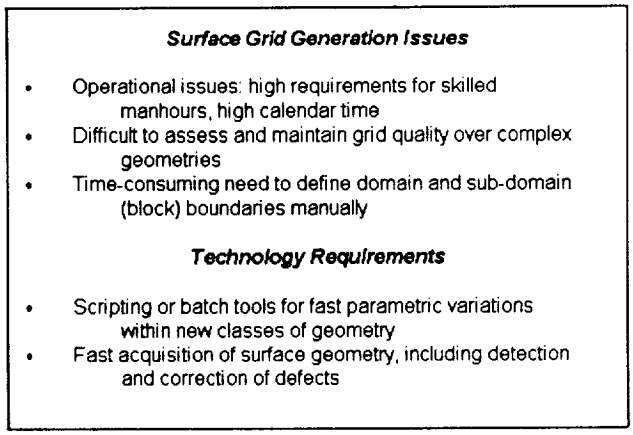



Figure 9 - Surface Grid Generation Issues

Surface grid generation, or a similar step to produce a satisfactory surface mapping, is a time-consuming step in the overall process. The quality of the surface grid has a large impact on the overall quality of the final analysis product. Our experience has shown that many methods do not ensure that the final surface grid points lie exactly on the original defined surface. Indeed, systematic variations have been noted, which can produce large-scale, erroneous, structures in the subsequent flow solutions. However, it is at present very difficult to assess surface grid quality (orthogonality, stretching, curvature, alignment) by any means other than visual inspection. Inspection, of course, is not a systematic process. The expertise and the sensitivities

of each inspector are different. Further, there are very few absolute measures of quality. This approach leaves a high probability that defects will not be detected at this stage, and they will remain in the surface grid to have a magnified impact in later steps of the process.

Volume Grid Generation - Volume grid generation is the process of filling a defined volume with a grid, using either structured or unstructured technology. This step of the process, by either technology route, is fairly mature today. As indicated above, the major issues are process issues (speed, reliability, versatility) rather than basic technical capability (see Figure 10).

***Volume Grid Generation Issues***

- Set grid density, stretching requirements manually in every block
- Scripting or batch tools for fast parametric variations within new classes of geometry
- Automated domain decomposition
- Improved control of orthogonality near walls
- Batch/Script tools for fast generation of surface grids in subdomains
  - Structured Grids
  - Unstructured Grids
- Default generation of volume grids within defined subdomains

Figure 10 - Issues in Volume Grid Generation

In the area of structured grids, several quality issues must be resolved. Elliptic methods are popular for the generality of their capabilities. However, these methods still have unresolved, systematic problems maintaining acceptable grid quality near both concave and convex corners. Convex corners invariably lose grid packing, while concave corners yield grid line crossovers and negative volumes. For unstructured grids, we lack systematic useful standards of quality - useful in terms of ability to represent the performance of the flow solvers without being excessively restrictive.

Block Boundaries - For multi-block structured grids, either patched or overset, the next step is to generate the block boundary coupling pointers. This is an identification of the grid point matchups between neighbor grids, for the purpose of coupling the flow solutions between adjacent grids. This is a key problem area for both types of structured grid technology. Since this issue falls somewhat ambiguously between grid technology and flow solver technology, often this issue is addressed inadequately. Avoidance of this problem is one of the major attractive features of unstructured grid technology (see Figure 11). Another quality issue is related to the placement of the overlapping boundary (high-gradient regions in the flowfield should be avoided - shocks, wakes, etc.). Further, cell sizes should be comparable in the two grids which are being coupled in the overlapping region.

***Block Boundary Issues***

- Difficult to assure quality in setting up coupled block boundary conditions - structured/patched, structured/overset, hybrid

Figure 11 - Block Boundary Issues

For patched (non-overlapping) grids, the problem is perhaps slightly easier since the coupling pointers are generated on two-dimensional surfaces (in the mathematical sense). For overset grids, the coupling occurs in a three-dimensional volume which is common to two or more grids. In either approach, the goal is to locate boundary points of one grid in terms of the mapping of the neighbor grid(s). Quality must be achieved, without any errors, in this process or the subsequent flow solution in all probability will be fatally compromised. This step of quality verification is time-consuming, though semi-automated procedures are available to assist the engineer effectively in this step of the process.

A similar problem may occur with unstructured grids, if the subsequent flow analyses are to be performed on a parallel processing computer system. For this application, a type of domain

decomposition must be accomplished on the full unstructured grid to establish subsets which will be passed to each processor. This decomposition can have a significant impact on the convergence rate of the composite solution and the processing time for each iteration. However, optimal characteristics of this decomposition are not fully understood at present.

## FUTURE STATE

An attractive future state can be identified, within the context of current structured and unstructured grid generation technology. This future state, I hope, can be achieved as a product of specific research into issues such as those which have already been noted. Strawman estimates of the time required to model a full aircraft configuration are noted for each step.

- The nominal geometry (i.e., before any modifications based on analysis goals) is generated in the CAD system. Either the CAD system or a separate procedure is used to identify defects, omissions, etc., in a timely manner, so that they are corrected before the geometry is used for any subsequent process. In the current process, these defects usually are to be corrected in the CAD process, which can take several days. With a more robust process, as discussed above, the required time should be greatly reduced.

- The geometry is modified, within the CAD environment, as appropriate for the subsequent analyses.

- The analytical tools for surface modeling and grid generation operate directly on the CAD surface models, or on a data format that is immediately derived without compromise from the CAD models. (Time for geometry acquisition: 2 minutes - a file transfer only).

- Semi-automated tools are used for surface grid generation and domain decomposition. These tools "suggest" default surface and block face grids, subject to approval by the engineer. Presumably, this process must take into account the goals of the analysis: parameters to be predicted, required precision, flow analysis code to be used. If the engineer chooses not to accept the suggestions, the same grid generation environment provides full tools, with high automation, to implement the engineer's desires. Quality of the final surface grid or mapping is verified by automated procedure at the end of this step. (Time to generate surface and block face grids: 4 hours).

- Overall parameters of the volume grid blocks - number of points, stretching functions, etc. - are set by a semi-automated process (automatic recommendations, with engineer having opportunity to modify the recommendations). This step, too, must take into account the goals of the analysis. Having set the overall parameters of the grid blocks, the actual grid is generated by a fully automated process. Quality of the grid is verified at the end of the process. (Time to generate volume grids: 1 hour)

- Block interface data is generated by a fully automated process. Quality of the interface data is verified at the end of the step. (Time to generate interface data - 10 minutes).

Each of these steps seems achievable over the next five years, with an appropriate research focus. It should be noted, the strawman process times to generate a complete multiblock grid for a complete aircraft, starting from a complete high-quality CAD definition, add up to less than six hours in this

vision of the future. Achieving this vision will be a major step toward providing the fast cycle time needed to support intensive use of analysis-based design for future aerospace vehicles. Of course, major improvements in calendar time for the flow solution and post-processing also will be needed.

Many technical communities in addition to CFD and the aero-propulsion community will benefit from this research. The surface grids produced by this process will be of value to many other technical communities in the design process that require definition of the exposed vehicle shapes.

## SUMMARY AND CONCLUSIONS

The process of surface modeling and grid generation is, at present, based on interactive (manual) operations from start to finish, often requiring some of the most highly skilled specialists in the analysis community. This technology fills the gap between the design community (based on CAD systems) and the analysis community. These specialists must be conversant in both sets of technology.

Some of the key, recurring problems in this area were mentioned previously. Several suggestions for future technology development have been identified. Another key issue has been mentioned, in various contexts, earlier in this paper. To maintain the highest quality in the products of surface modeling and grid generation, we first must be able to measure the quality. Metrics and tools are needed for a meaningful assessment of quality at every step of the process. A summary of these quality measurements needs is presented in Figure 12.

***Quality Measurement Requirements***

- Surface Models
- Surface Grid Quality
  - Including fidelity in conforming to the prescribed geometry
- Volume Grid Quality
  - Structured Grids
  - Unstructured Grids
- Block Boundary Quality
  - Boundary Condition Setup
  - Coupled Interfaces: Structured, Unstructured, Hybrid

Figure 12 - Quality Measurement Requirements

## ACKNOWLEDGMENTS

The information presented here is based on personal views of the author, and the views of the author's associates. In particular, this paper is based on observations and suggestions from Bob Bush, John Cooley, Greg Finfrock, Tim Gatzke, Walt LaBozzetta, and Todd Michal (all at McDonnell Douglas Aerospace, St. Louis).

## REFERENCES

1. Steinbrenner, J.P., J.R. Chawner, and C.L. Fouts, "The GRIDGEN 3D Multiple Block Grid Generation System," WRDC-TR-90-3022, July 1990.

2. Blake, M.W., P.A. Kerr, S.A. Thorp, and J.J. Chou, "NASA Geometry Data Exchange Specification for Computational Fluid Dynamics (NASA IGES)," NASA Reference Publication 1338, April 1994.

N95-28725

19951223O4

# REQUIREMENTS FOR EFFECTIVE USE OF CFD IN AEROSPACE DESIGN

Pradeep Raj*
Lockheed Aeronautical Systems Company
Marietta, Georgia 30063-0685

## SUMMARY

This paper presents a perspective on the requirements that Computational Fluid Dynamics (CFD) technology must meet for its effective use in aerospace design. General observations are made on current aerospace design practices and deficiencies are noted that must be rectified for the U.S. aerospace industry to maintain its leadership position in the global marketplace. In order to rectify deficiencies, industry is transitioning to an integrated product and process development (IPPD) environment and design processes are undergoing radical changes. The role of CFD in producing data that design teams need to support flight vehicle development is briefly discussed. An overview of the current state of the art in CFD is given to provide an assessment of strengths and weaknesses of the variety of methods currently available, or under development, to produce aerodynamic data. Effectiveness requirements are examined from a customer/supplier view point with design team as customer and CFD practitioner as supplier. Partnership between the design team and CFD team is identified as an essential requirement for effective use of CFD. Rapid turnaround, reliable accuracy, and affordability are offered as three key requirements that CFD community must address if CFD is to play its rightful role in supporting the IPPD design environment needed to produce high quality yet affordable designs.

## INTRODUCTION

Over the last thirty years, we have seen a phenomenal growth in speed and memory of digital computers with estimates ranging from three to four or more orders of magnitude. Scientists and engineers have successfully exploited this growth to significantly advance the frontiers of science and technology. In the present context, advances in our ability to model complex flow fields are of most interest. The corresponding enabling technology, widely known as computational fluid dynamics or CFD, is now an integral part of all science and engineering disciplines where fluid dynamic interactions play an important role. From a scientific view point, the critical importance of CFD is obvious from the role it is playing in providing a better understanding of the more complex flow physics in general and turbulence in particular (ref. 1, 2). From the engineering vantage point, CFD holds considerable promise to revolutionize the design of flight vehicles, automobiles, turbomachinery, etc., provided that its potential is successfully harnessed. Even a cursory glance at the ever growing list of technical publications documenting CFD applications should be enough to convince even skeptics of CFD's potential. It is impractical to include an exhaustive list of CFD publications in this paper; interested readers should consult Reference 3 for a representative sampling of the variety and complexity of geometries and flow fields that can be modeled using modern CFD techniques. However, it would be a mistake to consider the number and volume of publications on CFD as a testimony to its effective use in the aerospace design environment. With this basic premise, the present paper examines the issue of CFD effectiveness in aerospace design and identifies some of the key requirements that CFD must meet in order to be fully effective. The outcome of any examination is generally a function of the examiner's level of knowledge, past experiences and personal biases. The reader should be forewarned that the present effort is also subject to the same influences.

The remainder of the paper is organized along the following lines. The section on Design Process and Role of CFD immediately follows this Introduction section and contains author's observations on the general nature of the aerospace design process and where CFD fits in. Both conventional design practices and transformations taking place to accommodate the emerging integrated product and process development environment are considered. The following section provides an overview of the current state of the art in CFD and the direction it appears to be heading. Requirements for effective use of CFD are discussed in the next section. The paper then concludes with a few summary observations in the Concluding Remarks section.

---

*Technical Fellow

Copyright © 1995 by Lockheed Corporation. All rights reserved.

PRECEDING PAGE BLANK NOT FILMED

## DESIGN PROCESS AND ROLE OF CFD

In this section, some general observations are presented about current trends in aerospace design. What follows is not a comprehensive discussion of all relevant issues; such a detailed discussion is probably outside the scope of the paper and certainly beyond the limited abilities of the author. Instead, observations are presented mainly to help set the stage for discussing the role of CFD and partly for the sake of completeness. While the author fully recognizes subtle and sometimes not so subtle differences among the design processes of individual companies, it is hoped that what follows will faithfully represent important aspects of the current trends at a majority of companies and thereby provide a valid basis for the rest of the discussion. Readers are strongly encouraged to read many interesting and thought-provoking articles that have appeared in literature over the past few years including but not limited to a survey paper by Miranda (ref. 4) on application of CFD to airplane design, the Lanchester Memorial Lecture by Hancock (ref. 5) on the role of computer in aerodynamics, a paper by Miranda (ref. 6) on challenges and opportunities for CFD in fighter design, a paper by Cosner (ref. 7) on issues in aerospace application of CFD, and the Wright Brothers Lecture by Rubbert (ref. 8) on the role of CFD in the changing world of airplane design.

### Conventional Design Practices

A schematic of the aircraft design process, shown in Figure 1, forms the basis for general observations about the conventional design practices. The process is divided into three phases, (1) Conceptual, (2) Preliminary, and (3) Production, that are carried out in sequence. In the conceptual phase, a set of candidate configurations is defined that is expected to meet customer specifications and requirements. Following trade-off studies using estimates of performance, weight, cost, etc., a single configuration is selected for further development. The design typically undergoes numerous modifications during the conceptual and preliminary design phases. The goal is to create an "optimum" design that satisfies all customer requirements. In the production design phase, the final layout and more extensive validation are carried out prior to releasing the design for manufacturing. In each design phase, the myriad of activities that take place can be broadly placed into synthesis or analysis categories. Synthesis covers defining, refining, and altering concepts and configurations; analysis encompasses methods, tools and expertise to produce data and its use in evaluating concepts and configurations. Their roles are illustrated in Figure 2.

Although highly sophisticated tools and techniques have evolved to support synthesis and analysis activities, completing a design cycle with comprehensive and extensive evaluations of competing concepts and configurations can take several months and many, many labor hours. Most of the time and effort goes into generating data for different disciplines that design teams need to reach design closure. Many times, data from different disciplines create conflicting demands on the direction in which a configuration should be altered. Such conflicts cannot be easily reconciled without the timely availability of accurate quantitative information about the interdisciplinary relations among the design variables. A simple example is that of wing design. The "best" set of geometric parameters obtained from purely aerodynamic considerations may not look so good when structural integrity aspects are taken into account. The real challenge then is to guide the design in a direction that offers the "best" balance between aerodynamic and structural efficiencies. This can be accomplished only through a good understanding of the interrelationship of aerodynamic and structural design variables. Adding more disciplines such as producibility, manufacturability, maintainability, cost, etc., further compound the problem but they have to be taken into account before a design can be finalized. The same basic principles apply to the design of a complete aerospace system of which wing may be just one component. However, the challenge grows nearly exponentially with increasing complexity of the system. At present, procedures for generating quantitative data on interdisciplinary relationships are less than satisfactory at best, and nonexistent at worst. Design team's decisions are therefore highly dependent on the intuition and experience of its members, especially in the early stages of product development.

The large time and effort associated with a complete design cycle limits the number of cycles that can be conducted to explore a wider spectrum of alternatives within schedule and cost. It cannot be overemphasized that schedule and cost constraints are central to all industrial design processes; sometimes they get lost in discussions of advanced technologies. It should also be noted that decisions made in the early stages of design have far-reaching consequences for the life-cycle cost of the final design. It has been variously estimated that 70% to 90% of the life-cycle cost of an airplane is locked in during the conceptual and preliminary design phases. Tools and techniques used in the early phases are typically not as advanced or sophisticated as needed to produce highly accurate and reliable data. As the product evolves over time and data from more detailed investigations come in, design teams face the prospect of either changing the design at the expense of increased cost or retaining a design that may not meet all

customer requirements and specifications.

## Transition to IPPD Design Processes

It is obvious from the discussion above that the deficiencies in conventional design processes make it extremely challenging for design teams to produce *high quality* designs at *affordable* prices. The challenge is not either quality or affordability, but both simultaneously. Without successfully meeting this challenge, the U.S aerospace industry will have a difficult time in maintaining its leadership position in the increasingly competitive market place of the '90s and beyond. To address this concern, the industry and government jointly initiated many studies during the 1980s which led to a widely accepted conclusion that industry must transition to an integrated product and process development (IPPD) environment. IPPD is characterized by integration of all aspects of product development including design, marketing, manufacturing, and product support. The IPPD approach relies on considering *all* requirements and constraints from the start rather than altering a design in its later stages to facilitate manufacturing or accommodate product support needs. Proper trade-offs can therefore be made early and the need for design changes later on is considerably reduced. The result is improved quality and increased productivity of the entire development process.

In the IPPD context, design is viewed as an integrated multidisciplinary process. A key distinguishing feature of the integrated process is that it incorporates fast, accurate and cost-effective means of generating data for each contributing discipline as well as for complex interdisciplinary relationships among design variables. Availability of such data is critical to driving the design in the right direction. The integrated process must not be construed as an "automated design process." It cannot substitute for human creativity and unique synthesis ability. What it can do well is to shorten the design cycle time by expeditiously providing design teams with data needed to make more informed decisions and thereby alleviate the serious shortcomings of conventional design processes. Design teams can then devote more time and effort to considering a broader set of options with attendant improvements in quality and productivity. It must also be noted that the integrated process does not in any significant way differ from the conventional process in *what* activities are actually carried out, the significant difference is in *how*.

## Role of CFD

In the opinion of the author, the CFD technology will play a pivotal role in the implementation of the integrated design process and in its eventual success in improving quality and reducing cost of aerospace designs. Why? Because accurate estimation of aerodynamic data is essential to any flight vehicle design. Force and moment data are needed to evaluate performance and flying qualities; surface pressures provide inputs for structural design; and flow-field data facilitate systems integration, such as the integration of propulsion system with airframe. Using wind-tunnels alone to produce the desired aerodynamic data is too costly and time consuming to meet the basic requirements of the integrated process. A judicious mix of wind-tunnels and CFD is already beginning to pay off in design projects; the paper by Bangert et al (ref. 9) on F-22 tactical fighter design being a case in point. With continuing advances in CFD, there is ample reason to believe that an even stronger partnership with wind tunnels will emerge to produce aerodynamic data in a more timely and cost-effective manner.

Additionally, there are two areas where CFD can play an important role because it holds an edge over wind tunnels. First, CFD affords a means of computationally defining and/or refining geometric shapes to produce certain specified flow characteristics while satisfying some prescribed constraints; this is not feasible in a wind tunnel. Second, a combination of CFD and advanced computational methods from other disciplines, such as structures, controls, propulsion, etc., offers the only practical means of generating interdisciplinary relationships among design variables which are a cornerstone of the IPPD design process. However, full benefits of CFD can only be realized if we can use it effectively in the aerospace design processes. Before discussing the requirements for effectiveness, the current state of the art in CFD is briefly reviewed in the next section.

# CFD STATE OF THE ART

A variety of CFD codes are presently available to generate aerodynamic data for a given design. The codes can be broadly categorized into four levels shown in Figure 3. Basic characteristics of each level of codes are highlighted in this section. The categorization into four levels is based on a number of factors including the timeframe

of introduction of the methodology to the flight-vehicle design environment, the nature of mathematical formulation, and capabilities of the codes. The lowest level codes, introduced back in the mid to late 1960s, are now widely used and accepted; the highest-level codes, introduced more recently, are still struggling to find their place. It has long been known that the Level IV codes, based on the Navier-Stokes (N-S) equations, can in principle simulate nearly all flow phenomena of interest to aerospace community for which the continuum assumption is valid. (The Boltzmann equations based on the kinetic theory of gases need to be solved for modeling molecular flows; the related numerical techniques will not be covered in this overview.) However, adequate computer power and efficient numerical algorithms to solve the N-S equations were not available in the 1960s. This forced researchers to explore alternatives based on inviscid approximations to the N-S equations; the first three levels correspond to codes based on a hierarchy of inviscid approximations. Different mathematical formulations largely dictate the capabilities and limitations of the codes in modeling flow physics, and the associated numerical formulations have a strong bearing on the usability and applicability of the codes. In comparing the capabilities of different levels of codes, the focus is deliberately on complex geometries as they dominate the aerospace design landscape.

## Level I: Linear Potential Codes

The linear potential codes are based on the Prandtl-Glauert or Laplace equations which form the lowest level of inviscid approximation to the N-S equations. Most of the codes employ the boundary integral approach to solve the governing partial differential equations (PDEs). The equations along with the boundary conditions are cast in a surface-integral form using Green's theorem. The solution is constructed by discretizing the geometry into small elements and assigning a type of singularity (sources, doublets, or vortex filaments) to each element. The singularity strengths are determined by satisfying the no-normal-flow condition at a control point on each element. Depending upon the approximations used in surface discretization (mean surface or actual surface) and the type and functional form of singularities (constant source, constant doublet, linear doublet, etc.), codes with different characteristics (ref. 4) can be developed. The simplest codes, widely known as vortex-lattice methods, employ mean-surface representation of geometry and vortex-filament singularities, e.g., the VORLAX code (ref. 10). When the actual surface geometry is used, the methods are commonly referred to as panel methods. Low-order singularity distributions, constant on each element, have been employed in the QUADPAN code (ref. 11) and higher-order distributions, linear or quadratic functions, in the PANAIR code (ref. 12).

Although the simplicity of mathematical formulation of the linear potential codes inherently restricts their validity to purely subsonic and supersonic attached flows, they are quite extensively used in design efforts due to the ease of use, computational efficiency, and a high level of confidence built upon years of use. An experienced user can set up a computational model in a matter of hours even for relatively complex configurations like a complete aircraft. The computational times are small ranging from a few seconds on supercomputers to a few minutes on workstations. However, user expertise is crucial to ensure that results are correctly interpreted. The vortex-lattice methods and panel methods generally provide good estimates of lift, induced drag, moment coefficients and pressures for steady flight conditions. This data usually form the basis for performance and weight estimations in the early stages of design. Some of the codes also offer a design option that can be used to determine geometric characteristics (like twist and camber of a wing) for a prescribed set of aerodynamic parameters. To meet the aerodynamic data needs of the aeroelastic and flutter disciplines, versions of doublet-lattice method (ref. 13) are the codes of choice. The linear potential codes were first introduced into the aircraft design environment in the late 1960s and the entire class of codes reached a high level of maturity in the early '80s. With the possible exception of the oscillatory aerodynamic codes (ref. 14, 15), very little effort is presently going into research and development of this level of codes.

## Level II: Nonlinear Potential Codes

The nonlinear potential methods are based on either transonic small perturbation (TSP) equations or full-potential equations (FPE). Their ability to model transonic flows with shocks is the most significant benefit over the Level I codes. However, this benefit comes at the expense of added complexity stemming from the need to resort to a field approach to solve the nonlinear PDEs. The field approach requires that a region of the flow field surrounding a given configuration be divided into small elementary volumes; it is no longer enough to just divide the surface. In practice, TSP codes are easier to use than FPE codes, especially for complex geometries, because of the differences in boundary condition treatment. The TSP approach permits a simplified treatment based on the application of the no-normal-flow condition at a mean surface. In contrast, the FPE approach requires application to the actual surface. Consequently, Cartesian field-grid systems suffice for a TSP code whereas the FPE codes need boundary-conforming

grids. Cartesian grids are considerably easier to set up as compared to the boundary-conforming grids. Of course, the TSP codes suffer from limitations on the class of geometries and flow conditions that they can model accurately—a direct result of their simplified boundary-condition treatment.

Considerable progress was made throughout the 1970s towards developing a practical transonic-flow analysis capability based on FPE and TSP approaches. For steady-flow computations, the TSP code of Boppe (ref. 16) and the FLO-series of FPE codes of Jameson and Caughey (ref. 17) are representative examples. The promise and excitement of the newly-found ability of computing transonic flows were so strong that even wing design procedures (ref. 18) were developed while the analysis methods were still evolving. Reference 19 is a good source of additional details of progress made during the '70s. Since transonic flows are particularly susceptible to viscous effects associated with shock/boundary-layer interaction, considerable research was also done in coupling inviscid TSP and FPE codes with boundary-layer codes. In addition, an aeroelastic analysis capability based on the TSP formulation (ref. 20) was developed.

Although the Level II codes offered the much needed capability of modeling transonic flows, they did not find the same level of widespread acceptance as the Level I codes. A variety of factors contributed to this situation including the then level of grid generation technology which was not conducive to applying the codes on a regular basis to anything more complicated than wing or wing-body configurations, and the limited region of flight envelope (transonic cruise) where the codes could produce data of acceptable accuracy. Applications of the codes indicated, as one might have suspected, that solution accuracy deteriorated if the actual flow being modeled contained strong shock waves or large regions of vorticity (e.g., leading-edge vortices). Usefulness of the codes was therefore severely limited. For instance, they could not adequately handle a whole class of aerodynamic problems associated with fighter design. In the author's opinion, nonlinear potential codes were basically taken over by the rapid pace of advances in Euler codes in the early '80s. Development of the TRANAIR code (ref. 21) was an exception to this trend. TRANAIR adopts an unconventional hybrid approach combining the flexibility of panel methods to handle complex geometries with the ability of FPE formulations implemented on Cartesian grids to handle nonlinearities of transonic flows. Considerable success has been reported (ref. 3, chapters 15 and 19) in applying this code to aerospace design problems.

## Level III: Euler Codes

The Euler equations, which form the basis of Level III codes, represent the highest-level of inviscid approximation to the N-S equations. By permitting nonisentropic shocks and rotational flows to be part of the solution, Euler codes alleviate the major limitations of potential-flow methods albeit at the cost of additional computational expense. The added expense comes from the need to solve at least four and generally five coupled first-order PDEs instead of one second-order PDE. However, two factors at the dawn of the eighties convinced most researchers to shift their focus to Euler equations. These factors were: (1) projected growth in computer power, and (2) development of more efficient numerical algorithms to solve the Euler equations (ref. 22, 23). In addition, the accelerated pace of boundary-conforming grid generation technology combined with the use of finite-volume concept to decouple flow solvers from grid mappings held considerable promise for realizing CFDers dream of analyzing realistic geometries, such as a complete aircraft, on a regular basis. A synopsis of the impressive progress made so far is presented here; details can be found in many publications including Reference 3 and a recent AGARD report (ref. 24).

Two distinct development paths can be identified for Euler codes: one based on hexahedral structured grids and the other on tetrahedral unstructured grids. During the early part of the eighties, most researchers focused their energies on structured-grid methods whereas the interest shifted considerably towards unstructured-grid methods from mid-eighties onwards. This shift was prompted by the realization that unstructured grids afforded greater flexibility in handling complex geometries and promised to "automate" the grid-generation process. Structured-grid advocates pursued a multiblock strategy to overcome the difficulties encountered in handling complex geometries, and codes based on patched or overset multiblock grids evolved to a high degree of sophistication. In spite of many publications detailing the virtues of one approach over the other and considerable advances in grid-generation techniques, the fact remains that constructing multiblock grids for complex geometries continues to be a labor-intensive and time-consuming task and unstructured-grid generation is not yet sufficiently automated although it certainly requires less time and effort. The recent resurgence in Cartesian-grid methods (ref. 25, 26) offers an attractive alternative because it essentially dispenses with the difficulties of grid generation leading to a considerable reduction in time and effort of applying them.

Two other aspects of Euler code development deserve mention. First, shock-capturing rather than shock-fitting has become the preferred approach. Both upwind and adaptive-dissipation schemes have been employed to a great degree of success on all kinds of grid systems. (Although the battle between the advocates of each scheme rages on, intensity has gone down considerably compared to the early years.) Second, most codes solve time-dependent form of the Euler equations even for modeling steady flows. Convergence acceleration techniques, such as local time step and multigrid, are employed to obtain time-asymptotic steady-state solutions in a computationally efficient manner. Both explicit and implicit time-marching schemes have been effectively utilized. Due to the use of time-dependent equations, modeling of unsteady flows is relatively straightforward, and this aspect has been exploited to develop aeroelastic analysis methods (ref. 27). Recent attempts at developing inverse design (ref. 28) and aerodynamic design optimization (ref. 29) methodologies are also noteworthy; their progress is being carefully watched.

Euler codes give us a powerful tool to analyze configurations of interest throughout the subsonic to hypersonic flight regime. This, combined with their demonstrated ability to automatically capture rotational flow regions (such as wakes shed behind wings and vortices emanating from sharp, highly-swept leading edges of delta wings), requiring no explicit a priori definition of such regions, renders them significantly more useful than the Level I or II codes. They are also beginning to make inroads into supporting the data needs of airplane design but in later stages (ref. 9). However, the implications of neglecting viscosity should be clearly recognized. Whereas the Euler codes are superior to the nonlinear potential codes in modeling strong shocks, their solutions are not necessarily closer to the actual flow which is likely to exhibit the effects of shock-induced separation. Some researchers have combined Euler codes with boundary-layer codes to more accurately model transonic flows on wing and wing-body configurations. The Euler codes also have an edge over potential-flow methods in capturing leading-edge vortices. But the location and strength of the primary vortices may not be accurate in cases where the secondary and/or tertiary vortices exert considerable influence. Also, the codes cannot provide an estimate of total drag (including skin-friction) or model flow separation from smooth surfaces. It is, therefore, not surprising that development of N-S codes has been aggressively pursued in parallel.

## Level IV: Navier-Stokes Codes

Navier-Stokes codes have a great deal in common with Euler codes. In practice, a single code usually serves the need of solving both Euler and N-S equations. This follows directly from the similarities between the two sets of equations. Elimination of diffusion terms readily converts the N-S equations to the Euler equations; they both share a common set of convective terms. However, the practical implications of this seemingly minor difference are enormous. For example, size of the computational model grows considerably due to the need of accurately resolving the diffusion terms which require highly clustered grids close to solid surfaces (as well as in other regions where viscous stresses are large). This has a bearing on grid generation, numerical algorithms, and computational resources. With appropriate grid clustering, we can solve the N-S equations to simulate laminar flows in a relatively straightforward fashion. But using these equations to directly model even simple turbulent flows stretches the current supercomputers to their limits. At present, the Reynolds-averaged Navier-Stokes (RANS) equations are used almost exclusively to simulate complex turbulent flows. For a large majority of problems, the thin-layer approximation to the RANS equations is employed to reduce the problem to a manageable size. But these simplifications impose a heavy toll; we now require a turbulence model!

A variety of turbulence models have emerged in recent years ranging from relatively simple algebraic models to more sophisticated Reynolds-stress models. They have been implemented into various codes. Impressive results have been obtained using multiblock structured-grid methods, both patched (ref. 30) and overset (ref. 31). Unstructured-grid techniques are also advancing at an accelerated pace (ref. 32). Many competing approaches are evolving ranging from tetrahedral grids to hybrid grids (combining prismatic grids in close vicinity of configurations with tetrahedral or Cartesian meshes elsewhere). In general, experiences to date in modeling turbulent flows have produced rather mixed results. There have been many successes in using simple models for relatively complex flows and some failures in using the more sophisticated ones for relatively simple flows. Attempts to refine existing models and develop new, improved ones continue unabated. Considerable research effort is also being devoted to developing models for laminar to turbulent transition, another area of great significance.

While progress is being made on many fronts, CFD practitioners dilemma is quite clear. For the foreseeable future, they will have to use RANS methods for modeling engineering problems of interest in aerospace design. Yet, the accuracy and reliability of the solutions for turbulent flows will continue to be subject to the inadequacies of

turbulence models. The prospects of a universal model are rather bleak; capturing the complex nature of turbulence in its entirety into a model with a few free parameters is a long shot indeed. Nevertheless, Level IV codes will continue to find increasing use in the years to come not because the turbulence and transition modeling difficulties will be fully resolved but to meet specific engineering needs. There are enough problems where viscous effects dominate and they can be properly simulated only by solving the N-S equations. Internal flow problems (inlet, diffusers, nozzles, etc.) and high-lift systems (multi-element wings) are two prime examples. Probably the best rationale for continuing use of Level IV codes, in spite of their limitations, may be taken from Bradshaw (ref. 33):

> "...we cannot calculate all flows of engineering interest to engineering accuracy. However, the best modern methods allow *almost* all flows to be calculated to higher accuracy than the best-informed guess, which means that the methods are genuinely useful even if they cannot replace experiments."

This brief overview of Level IV codes will not be complete without mentioning the recent emergence of Digital Physics™ technology (ref. 34, 35), developed and marketed by Exa Corporation, Cambridge, Massachusetts. Developers have shown preliminary incompressible-flow results on two test cases, backward-facing step and cylinder, for which good correlation with measurements were obtained without any turbulence modeling! The technology is claimed to have a fundamental advantage over conventional RANS CFD codes because it is free from the artifice of discretization. Depending upon the success in extending the technology to compressible flows and additional demonstrations, Digital Physics™ could provide a very attractive means of circumventing the turbulence modeling problem altogether for engineering applications.

## REQUIREMENTS FOR EFFECTIVE USE OF CFD

Webster's New Collegiate Dictionary defines effective as "producing a desired effect." In the context of using CFD in an industrial setting, the important questions are: Whose desires? What is desired? It is instructive to look at the whole issue from a customer/supplier viewpoint. The customer in our case is the design team and suppliers are the CFD practitioners. Then, CFD use can be considered effective only if the desires and expectations of the design team are met. (Note that your use of CFD is effective if the design team calls upon you the next time they have a need!) Design teams need a variety of data ranging from integrated quantities like forces and moments to detailed flow features like shocks and vortices. Their natural desire is to obtain data of highest fidelity within schedule and cost. It follows that CFD use would be effective if the CFD team can produce the highest-fidelity data while meeting the schedule and cost milestones of the design team.

The customer/supplier viewpoint also simplifies the issue of requirements for effective CFD use. It forces the practitioners to look at CFD from the customer's angle—and the view turns out to be quite different. What we then find is that the customer wants good quality aerodynamic data to help him do his job better, at the time that fits his milestones, and at an affordable price; the customer does *not* want CFD per se. For example, engineers involved in performance estimation want drag polars, stability and control engineers want derivatives of force and moment coefficients, structural loads engineers want surface pressures or aerodynamic influence coefficient matrices. To engineers in the design team, the important things are engineering data, schedule, and cost. If CFD can provide them with data they need when they need it, then they consider the use of CFD to be effective. To them, CFD is just another tool. They do not always, nor should they be expected to, understand subtle differences among various techniques upon which the plethora of CFD methods are built. Many times, CFD practitioners have "oversold" CFD to customers without fully appreciating each others point of view. This lack of appreciation has led to rather unpleasant situations when CFD practitioners had difficulty in satisfying customer expectations.

For effective use of CFD in aerospace design environment, the most essential requirement is to have a partnership between the design team and the CFD team. Before embarking on any task, it is crucial for design teams to clearly define their data needs and the associated schedule and cost constraints. CFD teams should then devise appropriate strategies and define a set of feasible options. The two teams should jointly select the option that best fits the needs. Without such a partnership, a design team all by itself is most likely to select an option based on past experiences and least likely to take advantage of new advances in technology. By the same token, CFD teams are likely to resort to their favorite method to address every demand of data without fully analyzing the potential quality, schedule, and cost implications.

Having dispensed with the CFD effectiveness issue from the customer's viewpoint, let us examine the key requirements from the viewpoint of CFD suppliers or code developers. This aspect has received considerable attention in the past with notable contributions by Miranda (ref. 4), Bradley (ref. 3, Chapter 25), Cosner (ref. 7), and Rubbert (ref. 8). The cited articles contain opinions and observations of industry leaders from Lockheed, McDonnell-Douglas and Boeing. Although the articles appeared at different times over the span of over ten years, many common themes run through them. As far as the relationship between CFD effectiveness and code characteristics is concerned, all views can be condensed into the following expression due to Miranda:

$$effectiveness = quality \times acceptance$$

Here, quality refers to accuracy and realism of the solution, and acceptance includes usability, applicability, and affordability. This expression impresses upon CFD developers the importance of the simple fact that focusing on either quality or acceptance alone is not desirable; our approaches must enhance *both* simultaneously if we wish to increase the overall effectiveness of CFD in an aerospace design environment. The basic premise of this expression is as true today as it was when originally proposed. Combining this expression with the role of CFD in the aerospace design process discussed earlier, the author proposes three key requirements that CFD must meet in order to be effective in an aerospace design environment: rapid turnaround, reliable accuracy, and affordability. A word of caution is in order before we discuss each of them. All three must all be considered *together* and not separately in evaluating the effectiveness of a particular CFD methodology.

## Rapid Turnaround

The first and foremost requirement is rapid turnaround. "Minimizing calendar time" is identified by Bangert et al (ref. 9) as one of the primary requirements based upon their assessment of CFD applications to F-22 design. Turnaround is meant to cover the entire time it takes from the initial go-ahead to the final delivery of data to the customer. A typical CFD application process requires three steps: (1) Pre-processing or acquisition of geometry and setting up of a suitable computational model, (2) Running a flow solver, and (3) Post-processing or extraction of desired aerodynamic quantities by processing the flow-solver output and delivery of data to the customer. In order to reduce the total turnaround time, each step must be carried out with utmost efficiency. Of course, the level of the selected CFD code, i.e., Level I or Level IV, has a strong influence on turnaround time; the lower-level codes offer quick turnaround and the higher-level codes take longer. This is obviously not a desirable state of affairs when we compare the regions of relevance among different levels of CFD in a simplified two-parameter design space as shown in Figure 4. The large extent of the Euler and N-S regions clearly points to the potential of significant payoffs if their turnaround time can be made comparable to that of the lower-level codes.

At present, the higher-level codes are considerably slower than the lower-level codes in producing aerodynamic data. Days of geometry acquisition, weeks of grid-generation, hours of execution time on supercomputers, and days of time-consuming and labor-intensive postprocessing, all contribute to the present situation. Pre- and post-processing steps are the primary culprits when only few analyses are conducted using a single model. If a large number of runs are made on a single model, the total amount of computer time (in wall-clock hours or elapsed time) can be substantial and may even overwhelm the pre- and post-processing times. The most challenging situation arises when the configuration geometry undergoes changes and multiple analyses have to be performed for each variation. However, that is precisely what the IPPD design environment demands of CFD! An integrated design process that significantly reduces design-cycle time depends on methods that are fast. The current higher-level CFD methods are just not up to the challenge, except perhaps for component-level design for some limited region of the flight envelope. The challenge for the CFD community is clear: develop appropriate technologies and integrate them in a manner that brings the turnaround time for each analysis to a matter of minutes. The list of potential enabling technologies includes: streamlined interfaces to computer-aided design (CAD) systems based on standard data-exchange protocols; nearly automated grid generation; parallel processing of flow solver software; intelligent systems for data analysis and management, to name a few. Ongoing research and development, some of it reported in the proceedings of this workshop, gives considerable hope and encouragement to CFD practitioners that the target is achievable.

## Reliable Accuracy

Although reducing turnaround time is crucial, producing data of reliable accuracy is of equal importance. A solution of reliable accuracy is one that comes with a known and acceptable error band on all quantities of interest to the customer. As pointed out by Bangert et al (ref. 9), F-22 design team relies primarily on wind-tunnel data due to

the limitations of current CFD codes in modeling viscous effects, especially when applied to complex geometries and very large speed, altitude, and maneuver envelope. An interesting thing to note here is that wind tunnels are not necessarily the best tools to generate desired aerodynamic data. They have limitations of their own such as support and wall interference effects, scale effects, etc. But design teams have built-in confidence in data coming from tunnels which have been used for almost as long as aeronautics has been around. Wind-tunnel test teams know their customer as well as understand the limitations of their tool. They have developed elaborate procedures to compensate for most, if not all, sources of error in data. In contrast, CFD methods do not have the same limitations as wind tunnels; flow analysis can, in principle, be conducted for arbitrary flight conditions. But, in practice, CFD teams have a good deal of difficulty in attesting to the reliability of their data. The situation must be rectified because without reliable accuracy of CFD predictions, producing "optimum" designs in an IPPD environment will remain an elusive goal.

Accuracy of computed solutions has two components: numerical and physical. A solution may be considered accurate in a numerical sense if it shows little or no sensitivity to changes in grids as well as other numerical parameters related to the algorithm. (It is assumed that the code in question has been verified as to the adequacy of its numerical formulation in solving the governing equations.) At present, there are few, if any, *practical* means of estimating the effect of grid resolution, truncation error, numerical parameters such as dissipation and dispersion, etc. Schedule and cost constraints of a typical design effort do not permit extensive investigations to determine the optimal grids and parameters. CFD teams usually rely upon previous experience and expertise but the situation is not totally satisfactory. What is really needed is built-in means of quantifying the level of accuracy. The problem is admittedly difficult but a solution is urgently needed if CFD is to be utilized effectively in the IPPD environment. In combining CFD with methods from other disciplines to produce interdisciplinary relationships among design variables, an assessment of the level of accuracy and associated error bounds of the solutions is even more critical. Incorporation of solution-adaptive techniques based on truncation error and/or numerical dissipation is one possible approach to address the problem of estimating as well as minimizing numerical errors. Some of the approaches, such as unstructured grids or Cartesian grids, are inherently more suitable to addressing this aspect.

Even if a code produces a numerically accurate solution on a given model, it is not trivial to determine how well the solution stacks up against the *real* flow—a measure of the physical accuracy. Keep in mind that when CFD is used in a predictive mode in a design environment, CFD teams do not have the luxury of comparing results with other data to determine the level of accuracy! To date, CFD community has advocated and conducted extensive "validation" exercises to generate correlations that can be used to substantiate claimed levels of physical accuracy. In practice, we have been able to barely "calibrate" the codes for specific applications of interest. (See ref. 3, chapter 25 for definitions of validation and calibration.) Why? Because major difficulties arise in planning a comprehensive validation effort. For example, how many test cases, what combination of flow conditions for each test case, and what range of values for each condition must we consider before a code can be declared as fully validated? A matrix of runs using a reasonable set of test cases and conditions quickly grows into a monumental task. Even if we assume that adequate resources as well as measured data are available for carrying out such a task, we run against the tide of technology dynamics. Rapid pace of advances in hardware, numerical algorithms, and models of turbulence and transition fosters an environment where codes are never quite "finished." Sometimes the changes are nominal, many times not. Cost/benefit assessment of any plan of allocating huge resources to validate a code that might be superseded the next day by a "new and improved" method does not support the validation route. CFDers inevitably fall back upon calibration to meet the immediate needs for a class of problems of greatest interest. This situation is likely to persist as long as we rely on RANS codes that require turbulence and transition models. For most applications, a judicious mix of CFD and wind tunnels will be the most effective strategy. Experience shows that a properly calibrated code can go a long way in enhancing the overall cost-effectiveness of CFD in aerospace design.

## Affordability

The third and final requirement is that of affordability. Costs associated with CFD use include both labor and computing expenses. At present, labor expenses are mainly connected with pre- and post-processing steps. For higher-level CFD codes, the labor expenses are still beyond the acceptable range. For example, the use of structured-grid methods requires several person-weeks of pre-processing effort whereas a desirable value is closer to a few person-hours. Unstructured-grid (tetrahedral and Cartesian) methods appear to be quite promising in reducing the level of effort. Progress in developing streamlined interfaces between grid-generation methods and CAD systems is crucial to reducing the geometry acquisition time. These improvements will also help in evaluating design changes in an inexpensive manner. As a matter of fact, technologies needed to reduce labor hours are essentially identical to those mentioned earlier for reducing turnaround time.

Computing costs mainly relate to running the flow solver and may include grid generation in some instances. We typically need computers with high processing speeds and large memory. Without access to such machines, it is very difficult to produce the desired set of data on schedule. A typical design cycle can require hundreds of runs before a sufficient amount of data is generated. Since shortening the design cycle is one of the key objectives of the design teams, data must be generated over a matter of days and not months. Computing expenses to generate the desired data in this kind of timeframe must not be so large that the total product development cost will actually increase rather than decrease. Consequently, cost and computational efficiency of the entire hardware and software system are very important considerations for effective use of CFD. Strategies to increasing computational efficiency and reducing cost must be an integral part of all CFD development and applications planning.

## CONCLUDING REMARKS

CFD is a key enabling technology for the successful implementation of an IPPD environment needed for producing high quality yet affordable designs. Key requirements identified in the previous section must be addressed if CFD is to play its rightful role in the integrated multidisciplinary design process that is part of the emerging IPPD environment. The three key requirements are: *rapid turnaround, reliable accuracy,* and *affordability.* Unless they are met, the technology will not get *fully* incorporated into the industry design processes. Considering CFD as a tool—a means to an end—is necessary to evaluating and selecting the "right" technologies for building future capabilities. We must take a system-level approach to CFD; increasing the effectiveness of the overall CFD application process is more important to realizing the full benefits of CFD than enhancing the state of the art in some selected constituent elements. For example, development of a faster flow-solver will have the desired payoffs only if the associated pre- and post-processing tools are also speeded-up to permit a significant reduction in the overall turnaround time.

The challenge facing the CFD community today is to channel their efforts and resources in a manner that makes CFD fully responsive to the design needs. Numerous benefits will accrue from incorporating advanced CFD methods into the design processes. Using CFD methods that offer rapid turnaround capability will reduce design cycle time. Design teams can then explore a wider spectrum of alternatives within the schedule and cost constraints of a typical product development effort than is currently feasible. *Fast, accurate* and *affordable methods* will increase the productivity of the design process and reduce the number of expensive tests needed to support design data needs. The use of advanced methods may also reduce the number of cycles required for design closure. Design teams will be able to conduct extensive trade-offs needed to guide the evolution of a configuration in a direction that minimizes both acquisition and life-cycle costs. Improved understanding of component interactions will permit design changes to be made early and thereby reduce risk and increase the probability of meeting all customer requirements.

## REFERENCES


1. Landahl, M., "CFD and Turbulence," ICAS-90-0.1, The Daniel & Florence Guggenheim Memorial Lecture, 17th Congress of the International Council of the Aeronautical Sciences, Stockholm, Sweden, September 9-14, 1990.

2. Rai, M., Gatski, T., and Erlbacher, G., "Direct Simulation of Spatially Evolving Compressible Turbulent Boundary Layers, " AIAA-95-0583, 33rd Aerospace Sciences Meeting & Exhibit, Reno, Nevada, January 9-12, 1995.

3. "Applied Computational Aerodynamics," *Progress in Astronautics and Aeronautics*, Vol. 125, American Institute of Aeronautics and Astronautics, Washington DC, 1990, Henne, P.A. (ed.).

4. Miranda, L.R., "Application of Computational Aerodynamics to Airplane Design," *Journal of Aircraft*, Vol. 21, No. 6, 1984, pp 355-370.

5. Hancock, G.J., "Aerodynamics—the role of the computer," The 25th Lanchester Memorial Lecture, *Aeronautical Journal*, August/September 1985, pp. 269-279.

6. Miranda, L.R., "Transonics and Fighter Aircraft: Challenges and Opportunities for CFD," NASA CP 3020, Vol. I, Part 1, Transonic Symposium: Theory, Application, and Experiment, April 19-21, 1988, pp. 153-173.

7. Cosner, R.R., "Issues in Aerospace Applications of CFD Analysis," AIAA Paper 94-0464, 32nd Aerospace Sciences Meeting & Exhibit, Reno, Nevada, January 10-13, 1994.

8. Rubbert, P.E., "CFD and the Changing World of Airplane Design," ICAS-94-0.2, The Wright Brothers Lecture, 19th Congress of the International Council of the Aeronautical Sciences, Anaheim, California, USA, 18-23 September, 1994.

9. Bangert, L.H., Johnston, C.E., and Schoop, M.J., "CFD Applications in F-22 Design," AIAA Paper 93-3055, AIAA 24th Fluid Dynamics Conference, Orlando, Florida, July 6-9, 1993.

10. Miranda, L.R., Elliott, R.D., and Baker, W.M., "A Generalized Vortex Lattice Method for Subsonic and Supersonic Flow Applications," NASA-CR 2865, December 1977.

11. Youngren, H.H., Bouchard, E.E., Coopersmith, R.M., and Miranda, L.R.,"Comparison of Panel Method Formulations and Its Influence on the Development of QUADPAN, an Advanced Low-Order Panel Method," AIAA Paper 83-1827, Applied Aerodynamics Conference, Danvers, Massachusetts, July 1983.

12. Magnus, A.E., Ehlers, F.E., and Epton, M.A., "PANAIR - A Computer Program for Predicting Subsonic or Supersonic Linear Potential Flow About Arbitrary Configurations Using a Higher-Order Panel Method," NASA CR-3251, April 1980.

13. Giesing, J.P., Kalman, T.P., and Rodden, W.P., "Subsonic Steady and Oscillatory Aerodynamics for Multiple Interfering Wings and Bodies," *Journal of Aircraft*, Vol. 9, No. 10, 1972, pp 693-702.

14. Chen, P.C., Lee, H.W., and Liu, D.D., "Unsteady Subsonic Aerodynamics for Bodies and Wings with External Stores Including Wake Effects," *Journal of Aircraft*, Vol. 30, No. 5, 1993, pp 618-628.

15. Chen, P.C. and Liu, D.D., "Unsteady Supersonic Computations of Arbitrary Wing-Body Configurations Including External Stores," *Journal of Aircraft*, Vol. 27, No. 2, 1990, pp. 108-116.

16. Boppe, C., "Transonic Flowfield Analysis for Wing Fuselage Configurations," NASA CR 3243, 1980.

17. Caughey, D.A., and Jameson, A., "Progress in Finite-Volume Calculations for Wing-Fuselage Combinations," *AIAA Journal*, Vol. 18, No. 11, 1980, pp 1281-1288.

18. Hicks, R.M. and Henne, P.A., "Wing Design by Numerical Optimization," AIAA Paper 77-1247, 1977.

19. "Transonic Aerodynamics," *Progress in Astronautics and Aeronautics*, Vol. 81, American Institute of Aeronautics and Astronautics, Washington DC, 1982, Nixon, D. (ed.)

20. Borland, C.J. and Rizetta, D.P., "Transonic Unsteady Aerodynamics for Aeroelastic Applications, Vol. I—Technical Development Summary for XTRANS3S," AFWAL-TR-80-3107, 1982.

21. Samant, S.S., Bussoletti, J.E., Johnson, F.T., Burkhart, R.H., Everson, B.L., Melvin, R.G., Young, D.P., Erickson, L.L., Madson, M.D., and Woo, A.C., "TRANAIR: A Computer Code for Transonic Analysis of Arbitrary Configuration," AIAA Paper 87-0034, Aerospace Sciences Meeting, Reno, Nevada, January 1987.

22. Jameson, A., Schmidt, W. and Turkel, E., "Numerical Solution of the Euler Equations by Finite-Volume Methods Using Runge-Kutta Time-Stepping Schemes," AIAA Paper 81-1259, 1981.

23. Rizzi, A., "Damped Euler Equation Method to Compute Transonic Flow About Wing-Body Combinations," *AIAA Journal*, Vol. 20, No. 10, pp 1321-1328.

24. "Computational Aerodynamics Based on the Euler Equations," AGARD-AG-325, September 1994, Sloof, J.W. and Schmidt, W. (eds.)

25. Tidd, D.M., Strash, D.J., Epstein, B., Luntz, A., Nachshon, A., and Rubin, T., "Application of an Efficient 3-D Multigrid Euler Method (MGAERO) to Complete Aircraft Configurations," AIAA Paper 91-3236-CP, 1991.

26. Karman, S., "SPLITFLOW: A 3D Unstructured Cartesian/Prismatic Grid CFD Code for Complex Geometries," AIAA Paper 95-0343, 33rd Aerospace Sciences Meeting, Reno, Nevada, January 1995.

27. Guruswamy, G.P., "Unsteady Aerodynamics and Aeroelastic Calculations for Wings Using Euler Equations," *AIAA Journal*, Vol. 28, 1990, pp. 461-469.

28. Campbell, R.L. and Smith, L.A., "A Hybrid Algorithm for Transonic Airfoil and Wing Design," AIAA Paper 87-2552, 1987.

29. Reuther, J.J., Cliff, S.E., Hicks, R.M., and van Dam, C.P., "Practical Design Optimization of Wing/Body Configurations Using Euler Equation," AIAA Paper 92-2633, 1992.

30. Ghaffari, F., "Navier-Stokes, Flight, and Wind Tunnel Flow Analysis for the F/A-18 Aircraft," NASA Technical Paper 3478, December 1994.

31. Gomez, R.J. and Ma, E.C., "Validation of a Large Scale Chimera Grid System for the Space Shuttle Launch Vehicle," AIAA Paper 94-1859, CP 946, 1994.

32. Venkatakrishnan, V., "A Perspective on Unstructured Grid Flow Solvers," AIAA Paper 95-0667, 1995.

33. Bradshaw, P., "Turbulent Secondary Flows," *Annual Review of Fluid Mechanics*, Vol. 19, 1987, pp. 53-74.

34. "Digital Physics," in *Machine Design*, Vol. 66, No. 23, December 1994, pp. 96-100.

35. Teixeira, C., "Continuum Limit of Lattice Gas Fluid Dynamics," PhD. Thesis, Massachusetts Institute of Technology, Cambridge, MA, September 1992.

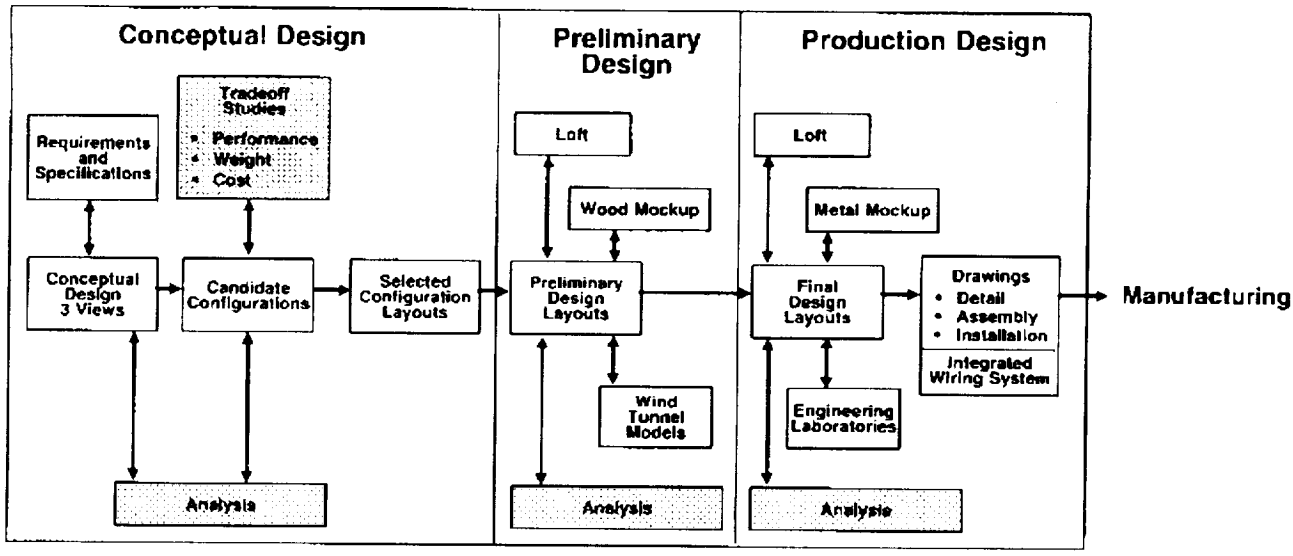



**Figure 1.—Schematic of conventional aircraft design process**

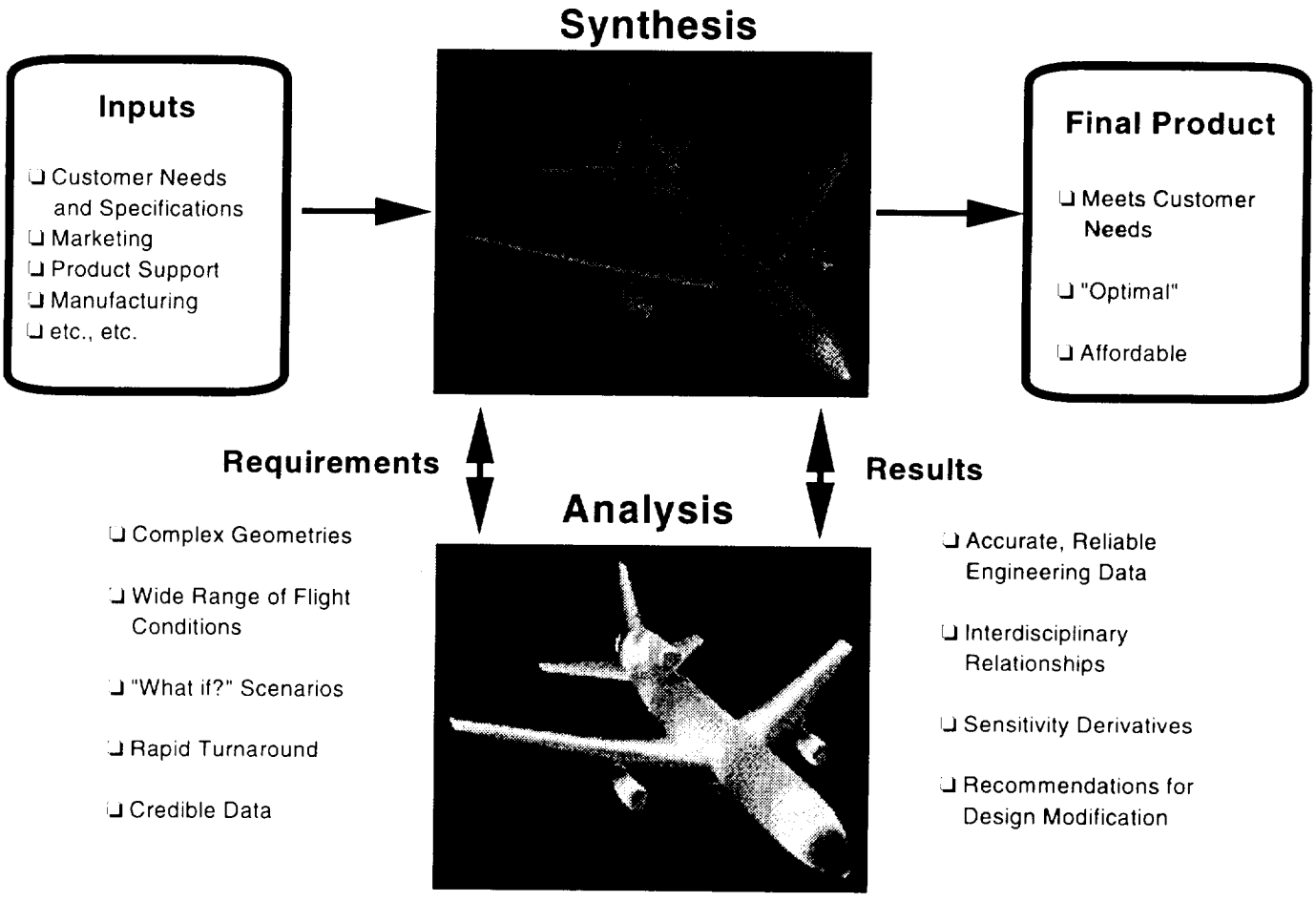



**Figure 2.—Role of synthesis and analysis in aerospace product development**

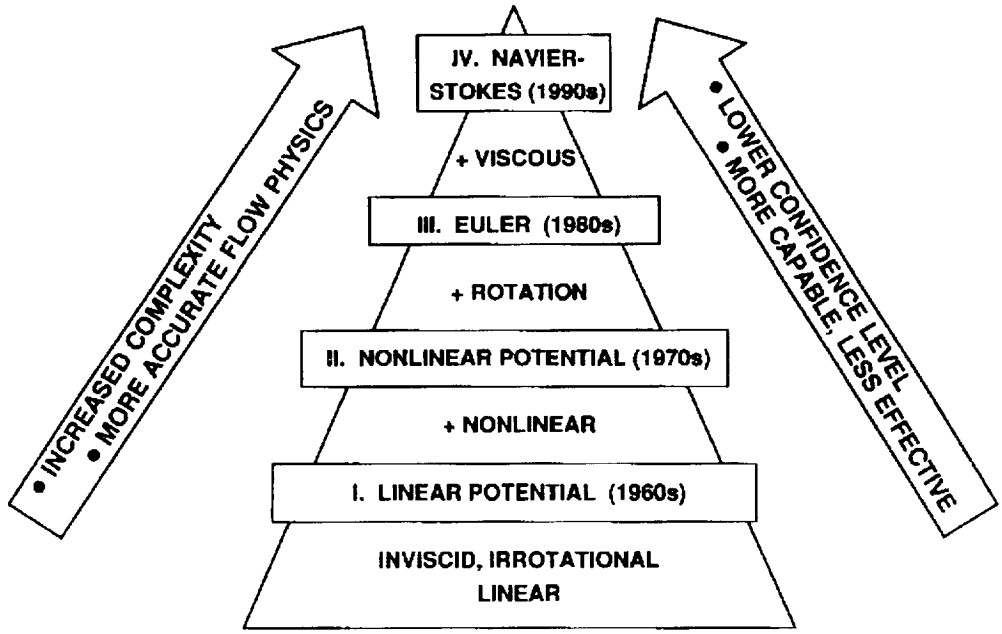



**Figure 3.—Four levels of CFD**

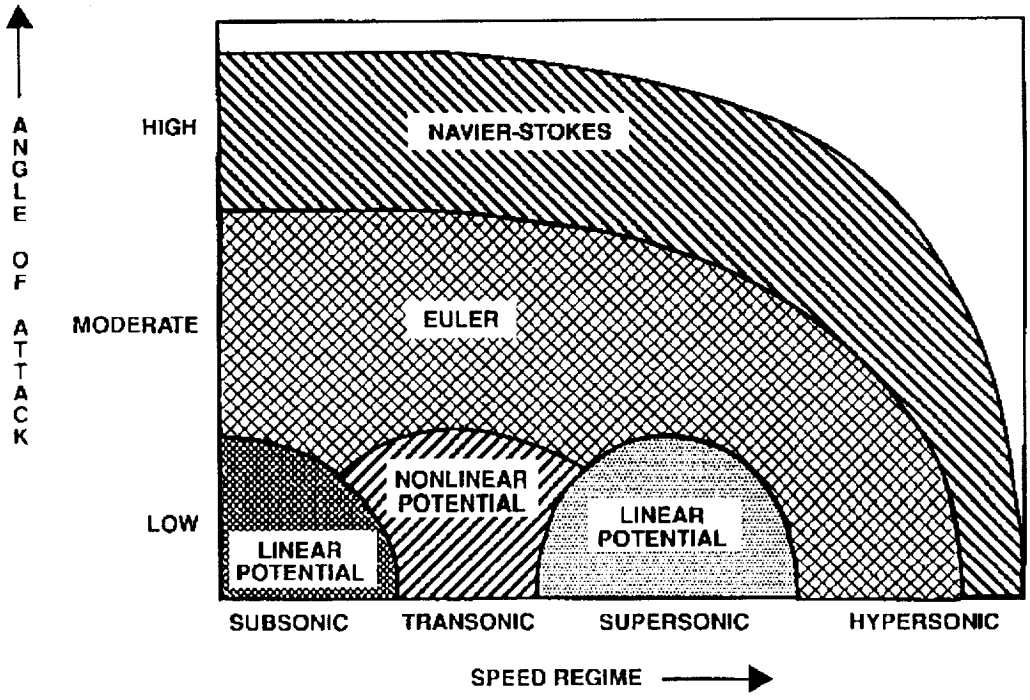



**Figure 4.—Regions of relevant CFD levels in a two-parameter design space**

19951122305

N95-28726

# GRID GENERATION AND SURFACE MODELING FOR CFD

Stuart D. Connell, Janet S. Sober and Scott H. Lamson
GE Research and Development Center
Schenectady, NY 12301

## SUMMARY

When computing the flow around complex three dimensional configurations, the generation of the mesh is the most time consuming part of any calculation. With some meshing technologies this can take of the order of a man month or more. The requirement for a number of design iterations coupled with ever decreasing time allocated for design leads to the need for a significant acceleration of this process. Of the two competing approaches, block-structured and unstructured, only the unstructured approach will allow fully automatic mesh generation directly from a CAD model. Using this approach coupled with the techniques described in this paper, it is possible to reduce the mesh generation time from man months to a few hours on a workstation.

The desire to closely couple a CFD code with a design or optimization algorithm requires that the changes to the geometry be performed quickly and in a smooth manner. This need for smoothness necessitates the use of Bezier polynomials in place of the more usual NURBS or cubic splines. A two dimensional Bezier polynomial based design system is described.

## 1 INTRODUCTION

The aerodynamic design of components is by and large achieved through "design by analysis". An engineer will begin with an approximation to the shape of a part. This is then analyzed utilizing a suitable CFD tool. Based on these results, changes to the geometry are postulated which will improve the part subject to certain constraints. These improvements, for example, may aim to increase the efficiency or decrease the manufacturing cost of the part. Having made the changes, the new design is reanalyzed and the results evaluated. This process may be repeated a large number of times during the design process. In general a limited amount of time and money are allocated for the design of a given component, hence the CFD code must allow a number of design iterations within these restrictions. To achieve this geometrical approximations are made and certain features are ignored because it is too costly to include them in the simulation. For some complex three dimensional configurations where geometrical simplifications cannot be made the use of a wind tunnel proves to be more cost effective.

With many currently available CFD technologies there is little room for improvement in component design within the constraints of time and money. Further improvements can only be achieved by including more geometric fidelity, reducing turnaround times and improving the physical models within the simulation. In this paper methods for improving the former two items will be investigated.

Many impressive calculations have been performed on complex three dimensional geometries employing block-structured techniques. However turnaround time makes these calculations impractical for routine design work. In some cases it may be possible to use these techniques in a design environment if mesh generation software is tailored for one particular topology [1]. This approach however requires a significant investment of time and money to develop such software which makes it too costly for a large number of problems. In addition

any design change that results in a change in topology would require much rewriting of the mesh generation software.

The use of an unstructured mesh presents a possible solution. Here mesh generation can be made automatic for arbritrary geometries thereby significantly reducing turnaround times. However the geometry specification (also common to the structured approach) and the generation of a mesh suitable for a viscous calculation still represent formidable tasks. These areas are addressed in detail in sections 2, 3 and 4.

Of the available unstructured mesh generators there is no accepted "best" method. It is possible using any of these methods to generate a mesh for complex configurations. The relative merits of the competing unstructured and structured approaches together with examples are presented in Section 2.

Within industry there is a strong trend towards representing all geometries by one single CAD model which is used and modified by a variety of disciplines during the design process. The model is ultimately used to manufacture the part. This approach has many advantages, for example, the elimination of the need for conversion programs, and their associated errors, to move between one geometry format and another. With powerful CAD packages available there seems little point in developing "in-house" software to represent and manipulate geometry. The obvious next step is to use these CAD models as the basis for a CFD calculation. Any geometry modification required by the design process can be efficiently performed using the CAD package. In section 3 the approach used to produce a computational mesh directly from a CAD model is described.

The desire to perform viscous calculations adds further constraints to the mesh generation algorithm. For these calculations high aspect ratio cells aligned with the flow gradient are required in the boundary layer and wake regions. Existing unstructured mesh generation algorithms aim to produce tetrahedra close to equilateral, or at best provide a limited ability to stretch elements. The use of equilateral cells in viscous regions would result in a prohibitively expensive algorithm. In addition there will ultimately be a requirement to adapt the mesh in viscous regions. To be efficient this adaption must be performed in an anisotropic manner, i.e. increasing the resolution normal to the wall while maintaining the streamwise spacing. The generation of meshes meeting the above requirements is the subject of much current research. An "inflation" technique to efficiently produce a near wall prismatic mesh about complex geometries is described in Section 4.

Unstructured meshes lend themselves well to adaption by reinforcement where new nodes are added into an existing mesh. In addition it is possible to use the refinement levels as the basis of a multigrid algorithm. Such a scheme is described and compared to other multigrid schemes in section 5.

The exact mathematical formulation of the surfaces used in the CAD model of the component to be meshed has been ignored and left to the CAD program. When it is desired to automate a CFD based design system the mathematical form of these surfaces becomes important. The use of Bezier polynomials as a basis for such a system is described in section 6.

Finally in section 7 recommendations are made for an ideal CFD system. These recommendations include the use of available technologies and the development of new ones.

## 2 MESH GENERATION

One of the aims when developing mesh generation software is to minimize the cost, both machine and human, of generating a mesh. As a designer's time is more valuable than CPU cycles this inevitably means minimising the amount of user interaction required to produce a mesh. In two dimensions mesh generation is essentially a solved problem. There are a variety of structured and unstructured algorithms which will produce a high quality mesh around complex geometries. For example the unstructured method described in [2] requires a few minutes of CPU time on a workstation and just four user inputs to generate a mesh: the file name for the geometry, the maximum element size, the minimum element size and a boundary curvature sensitivity. The procedure begins by placing points on the boundary to reflect surface curvature. These points are triangulated using a Delaunay

procedure. New nodes are now repeatedly inserted into the mesh to remove poor quality elements. The process is terminated when all elements satisfy the required quality measure. Figure 1 shows a mesh generated by this approach.

In three dimensions automatic mesh generation for complex geometries represents a formidable, though not intractable task. The block structured approach requires the specification of an initial blocking topology prior to generating the mesh in the individual blocks. The specification of the initial blocking topology is a global operation requiring knowledge of all the boundary surfaces. For the human brain this is a trivial task in two dimensions requiring a few mouse clicks in a suitable user interface to specify the block topology. The problem is significantly more difficult in three dimensions. Progress has been made in reducing the level of user interaction [3,4] to some degree, but due to the need for complex global operations it is unlikely that this will ever be fully automated.

In contrast the unstructured approach can be made automatic for arbitrary geometries. With these approaches there is no need for complex global operations. All operations are simple and local, requiring only local information. This fact alone makes it possible to automate the procedure.

There are three basic techniques used for automatic unstructured mesh generation:

(1) The octree approach [5,6] is based on successive subdivision of the domain to produce a Cartesian mesh. At boundaries many compute-intensive line surface intersection operations are required. This results in a high quality mesh in the interior with the worst mesh at boundaries. Smoothing operations are then used in an attempt to improve the near-wall mesh. Octree approaches do not lend themselves well to the generation of high aspect ratio cells needed for viscous simulations which will be described in Section 4.

(2) The Advancing front algorithm[7,8] is a two step procedure. First each surface is triangulated. This is done by placing nodes around the boundary of each surface in the model. These nodes are connected by edges to form an initial two dimensional front, this front is then advanced into the surface by building triangles on each edge of the front. In the second stage the triangulated surfaces form an initial front for the volume mesh. Using a similar algorithm to that used for the surfaces the domain is filled by recursively building tetrahedra on each face of the front.

(3) Delaunay based methods as described in [9,10,11] are the natural extension of the two dimensional algorithm described in [2] into three dimensions. As with the advancing front algorithm a surface triangulation is first generated and a constrained tetrahedralization is formed of these boundary triangles. The tetrahedral mesh is now generated by repeatedly inserting points into this mesh.

Of the above algorithms the advancing front and Delaunay methods have been found to produce smoother and more regular meshes than the somewhat irregular meshes typically produced by octree approaches. A comparison between the surface meshes produced by the advancing front and octree algorithms for a nacelle is shown in figure 2. In addition the formation of an initial surface mesh provides a natural framework for the viscous mesh generation procedure described in Section 4. Delaunay based methods have also been found to be significantly faster than the advancing front and octree methods.

The current drawback of the advancing front and Delaunay methods is the need to provide a background mesh. This background mesh is required to specify the mesh density throughout the flow domain. The automatic generation of this file, using for example surface curvature to drive the placement of source terms, would further accelerate the mesh generation process. An alternative approach suitable for the Delaunay algorithms is to abandon the the background mesh altogether. The surface mesh can be generated as in 2D [2] with the additional constraint that points be inserted into the mesh to resolve surface curvature. The 3D tetrahedral mesh can then be generated in a similar manner again by inserting points based on element quality.

Another point to note here is that much of the speed and ease of use of an automatic unstructured mesh generator can quickly be destroyed by the addition of a graphical user interface (GUI). These GUIs are of little or no use to a designer who needs to repeatedly mesh and solve on a series of similar geometries. Having to

repeatedly enter similar data is error prone and significantly hinders the mesh generation process. Much of the effort expended in writing such interfaces would be better utilized in further automation of the mesh generation procedure. Ideally any graphics in a mesh generator should be used for viewing the mesh and not for interactively generating the mesh.

## 3 LINK TO CAD SYSTEM

Computer Aided Design (CAD) systems can represent a component in three forms: solid, surface and wireframe models. A solid model contains information at two levels, topology and geometry. The topological entities are vertex, edge and face. Each face is surrounded by a number of edges, and a vertex lies at the ends of each edge. For example a solid model of a cube would comprise six faces, 12 edges and 8 vertices. The underlying geometrical entities used to define the actual shape of the object are referred to as point, curve and surface. These can be of various types within the model, for example, the geometry for a surface could be based on Non-Uniform Rational b-splines (NURBS) or Bezier polynomials. A surface model merely contains geometrical information for the individual surfaces. All topological information on how the surfaces fit together is lost. In addition there is no requirement that the bounding surfaces be closed. Wireframe models are a further simplification which only contain information on the outlines of the surfaces.

The use of a solid model to represent components is becoming more and more prevalent throughout industry, and are utilized by all disciplines from design to manufacturing. The models contain enough information to manufacture the part and hence provide a logical starting point for mesh generation. Generally with these models the solid part is the body, i.e. the metal, whereas to be of use for a CFD calculation the model needs to be inverted so the solid is the gas path around the metal. Fortunately this a relatively straightforward procedure within a CAD system.

An alternative approach is to base the mesh generation on a common surface model format such as IGES (or NIGES). This approach initially appears attractive as CAD systems can output IGES files, hence if a mesh generator can read IGES files it can work with all CAD systems. However with the surface model the topological information required to automatically build the mesh is lost and must be input by the user during the mesh generation process. Hence mesh generation cannot be made automatic and much of the advantage and elegance of unstructured mesh generation is immediately destroyed. In addition this approach necessitates the development and support of computer codes employing complex Graphical User Interfaces (GUI) to aid in the input of this information.

The strategy being developed at GE CRD is to generate an unstructured mesh by interrogating the solid model directly via the vendor supplied subroutine library in order to obtain information necessary to build the mesh. This is done as the subroutine package and solid model form a self consistent entity. Converting to some intermediate format such as some as yet undefined extension to IGES opens up many issues, for example tolerancing; while the model may be a perfectly valid solid model in one CAD package, it may fail some tests on the common file format. Coupling unstructured mesh generation with the native read of the solid model allows the mesh to be built automatically from the solid model with no further intervention from the user. At GE, solid models from the PARASOLID [12] based UNIGRAPHICS CAD system are used.

Unstructured mesh generators typically require a limited number of low level geometry and topology interrogation operations in order to build the mesh. It would be extremely useful if all the CAD vendors agreed on a standard in order to extract this information. Calls to these standard routines could then be made from the grid generation code thus allowing meshes to be generated from solid models from a variety of vendors.

As a standard set of geometry interrogation routines is not currently available an approach in a similar vein has been adopted. A set of low level interrogation routines to be used by the mesh generation package has been defined. Within these subroutines are the calls to the vendor specific routines which provide the required information. Examples of the sort of topological and geometric functionality needed are:

1) The total number of faces, edges and vertices
2) The (x,y,z) coordinates from face id number and parametric coordinates (u,v)
3) The id number of edges surrounding a face
4) The nearest point on a face to a given (x,y,z) location

Figure 3 shows the surface mesh for a three dimensional geometry generated directly from a solid model in 10 minutes on an HP735 workstation using a modified version of the advancing front program of [7].

## 4 VISCOUS MESH GENERATION

The requirements for a viscous mesh differ significantly from an inviscid mesh as it is no longer sufficient to fill the domain with equilateral tetrahedra. For efficient simulations high aspect ratio elements must be generated and aligned with viscous gradients. A recent paper [13] demonstrated such a capability in two dimensions. Here a layer of structured quadrilateral cells was wrapped around the body and extended into wake regions. The interior was then filled with triangular cells. An example of such a mesh is shown in figure 4. The quadrilateral mesh gives very efficient resolution of boundary layers and wakes. In addition it permits the directional refinement necessary in viscous regions. This is illustrated in figure 5. A natural extension of this approach to three dimensions is to first place structured layers of triangular based prismatic cells on an existing surface triangulation. This process can be thought of as "inflating" the surface triangulation. The interior is then filled with a tetrahedral mesh. The prismatic cells can have a high aspect ratio which will permit efficient resolution of the boundary layers. They also possess quadrilateral faces which will ultimately permit the directional refinement illustrated in figure 5.

There are a variety of methods currently being developed to produce near-wall prismatic meshes [14,15,16,17,18]. All the methods begin with a surface triangulation which is then marched or inflated towards the interior in a series of steps.

In [15] this marching is achieved through representing the initial surface triangulation by a number of non-intersecting hexahedral elements (voxels). The triangulation is contained within these voxels. The outer surface of these voxels is then smoothed to form the first inflated surface. Computing the intersection of normals from the original surface with the inflated surface forms the first prismatic layer. The process is then repeated to form the complete prismatic mesh. This method has the disadvantage (also common to Octree based methods) that the mesh will change if the geometry is rotated with respect to the coordinate system.

The unstructured hyperbolic mesh generation technique [16] was also investigated. This method was found to be prone to crossovers at sharp internal and external corners unless many explicit steps of the algorithm were taken. For some relatively simple test cases even with many steps a valid mesh remained impossible to obtain. With this method there is also little control over the spacing away from the wall. The specification of this spacing is crucial for turbulent calculations.

The advancing layers method [17,18] inflates the surface along quasi-normal directions with a modified advancing front type algorithm. When certain geometrical criteria are satisfied, such as distance from a wall or element quality, the algorithm reverts to the standard method. Numerical experiments indicate that this method can be made less prone to crossovers at sharp corners than the hyperbolic technique. It also has the advantage that near-wall spacing may be specified directly for turbulent flows.

The algorithm described in [14] also uses quasi-normal directions as a starting point for generating the prismatic mesh. New nodes are placed along the normal directions to form a structured mesh of prismatic elements which wrap around the viscous surfaces. The volume mesh generator uses this inflated triangulation as its initial front rather than the initial triangulation. This algorithm will now be described in detail.

To describe the combined algorithm of surface inflation and advancing front, a rectangular box with a cylindrical hole will be used as an example. This geometry is illustrated in figure 6. The cylindrical surface and lower wall are marked as viscous and are to be inflated. All other boundaries are either inlet surfaces, exit surfaces or inviscid walls and will not be inflated.

The procedure begins by triangulating the viscous walls using the surface meshing part of the advancing front algorithm. This surface mesh is shown in figure 7(a). The next stage in the procedure is to compute a quasi-normal direction at each node on the surface triangulation. The algorithm used to compute these normal directions is described in [14]. The initial surface is now inflated a specified distance $\delta$ along these quasi-normal directions. This gives the first layer of prismatic cells. This inflation process is repeated a number of times using different values of $\delta$ but the same normal direction. Typically $\delta$ varies between one layer and the next by a fixed geometrical factor ranging between 1.0 for a uniform mesh and 5.0 for a highly stretched mesh. The resultant prismatic mesh is shown in figure 7(b). The total thickness of this prismatic region is adjusted to encompass the expected boundary layer.

The remaining non-viscous surfaces are now triangulated. Any new nodes and edges generated by the inflation algorithm which lie on these surfaces are required to form part of the initial front for these surfaces. The newly triangulated surfaces and the inflated triangulation are combined to form a closed front. This becomes the initial front for the interior mesh generation algorithm. The resultant combined mesh is shown in figure 7(c).

For many cases this scheme is found to work well. However for some complex geometries a suitable algorithm to produce a set of normals, which in turn lead to a valid prismatic mesh of acceptable quality remains elusive. In addition, for boundaries which are close to one another, it may be possible for the prismatic meshes to overlap. To produce a robust mesh generator a fallback position is adopted. After the prismatic mesh is generated a number of checks for cell quality are made. These checks ensure that:

1) Each prismatic cell has positive volume,
2) No cell intersects any other and
3) Cells are of reasonable quality.

Any cell failing these tests is tagged for deletion from the mesh.

When cells are removed triangular and quadrilateral faces of elements below the inflated surface are exposed. To form a front the quadrilateral faces are divided into triangles. It is possible for these triangular faces to posess a high aspect ratio, especially near the wall. A front containing such faces has proven problematical to the volume mesh generator. The algorithm for removing these high aspect ratio faces is described in [14].

Meshes generated by this algorithm are shown in figure 8 for a turbine blade and figure 9 for a wing/pylon/nacelle. The mesh is illustrated by making various cuts through the prismatic region. As can be seen the mesh is of high quality and provides a good starting point for a viscous calculation.

Meshes generated by the above approach will contain a variety of element types. While it is possible to divide these elements and produce a purely tetrahedral mesh this is not the most efficient method. Modifying the flow solver to work directly on a mixed mesh is a better option.

## 5 ADAPTION AND MULTIGRID

Adaption forms an integral part of any unstructured CFD calculation, since using an unstructured mesh without adaption utilizes only half the power of the method. The use of the various meshes generated by an adaptive procedure in a multigrid algorithm is an obvious step and has been demonstrated in [19,20,21,22]. There are two basic methods used for adaption: reinforcement, where new nodes are added into an existing mesh, and remeshing where the entire geometry is remeshed.

Multigrid schemes based on adaptive remeshing have three disadvantages. The first is that the mesh generation algorithm needs to be robust and fast. If the scheme is not fast, the mesh generation time can quickly exceed the solution time and a lot of the benefit of multigrid is lost. The second disadvantage relates to the difficulty in computing transfer operators for the unstructured mesh. As the meshes bear no relation to each other this is a non trivial task. Complex data structures are required to avoid an $O(n^2)$ search. Finally, unless the meshes are generated through solution adaption then a large percentage of the nodes on the finer meshes may serve no useful purpose.

The multigrid scheme presented in [19] overcomes the above disadvantages. The process begins with an initial coarse grid. An initial solution is obtained on this grid. This solution is then examined to determine if the grid is sufficiently fine to resolve features of interest. In regions where more resolution is required the mesh is enhanced through reinforcement. A solution is obtained on this new grid. The solve and refine process is repeated until some desired level of accuracy is achieved. When using this approach the multigrid levels are formed naturally by the refinement procedure. The construction of the transfer operators is a simple task requiring a single pass through the data. Full details of the multigrid scheme are presented in [19].

The reinforcement procedure can be significantly faster than remeshing. Typically, the reinforcement procedure generates tetrahedra at the rate of 25,000 per cpu second on an HP735 workstation. In contrast the advancing front algorithm of [7] produces tetrahedra at the rate of 66 per cpu second. The newer Delaunay based meshing algorithms [9] are significantly faster than advancing front though still slower than reinforcement.

The initial grid for the reinforced procedure meshes has to be fine enough to resolve all the features of the geometry. If this is not so, it is possible for the node snapping procedure to produce a crossed over mesh, as illustrated in figure 10. This restriction prevents the full benefit of multigrid being realized on complex geometries as a relatively fine initial mesh has to be used. A possible solution would be to combine multigrid by agglomeration [23] with the reinforcement based multigrid. Coarse multigrid levels could then be generated from the initial mesh by agglomeration and finer levels by reinforcement.

## 6 SURFACE MODELING

More complex geometric models of internal and external systems require a greater degree of rigor in geometric definition. This requirement together with pressure for standardization and cost effective engineering are driving design of aerodynamic devices to be either based on standard CAD systems or to produce geometric models compatible with those packages. By basing the analysis on generic CAD systems, it becomes feasible to analyze aerodynamic devices with increased detail included, such as wings with flaps deployed and nacelle/pylons attached instead of analyzing an idealized wing. Similarly, turbine blade models may include shrouds, cooling holes, and endwall gaps previously ignored. With increasing focus on system level integration and optimization, all engineering disciplines require access to a consistent geometric model, often refered to as a master model.

The geometric model for aerodynamic devices should provide good support not merely for representing the final geometry but for the process of reaching that design. Aerodynamic surfaces are frequently free form sculptured surfaces. While low order polynomial (cubic) splines (or cubic NURBS in 3D) are quite suitable to represent geometry of this type, they provide poor support for the design process [24,25,26,27]. As a designer (either human or an optimizer) modifies a single cubic spline control point to find a better design, the surface curvature develops large oscillations undesirable to the fluid flow (see figure 11). It has been found that using high order Bezier curves instead of cubic splines has a much more desirable response to control point manipulation, as shown in figure 12. This technology has been developed into a quasi-3D turbine blade design tool which finds application in both interactive design of blades and automated design where it is driven by an optimization system [28].

For the master model concept and automated grid generation to deliver the required benefits in streamlining the engineering process, the geometric processing required must proceed smoothly and reliably. Unfortunately, the current state of commercial solid modelling software has not yet attained this level of robustness. Automation and speed of the design process are dependent on the correct reliable performance of solid modelling software.

Particularly in this age of virtual corporations, outsourcing, and strategic alliances, it cannot be expected that a single CAD system will be selected as the universal supplier of geometry. While it is possible to set up a standardized interface to geometry packages that grid generation software could use, as described in section 3, experience in two-dimensions indicates that once you go beyond static CFD solutions and include analytic sensitivites, design optimization or inverse design, it is required that the CFD tools have an explicit geometric capability built in to them. It is quite possible to have a generic geometric interface for static geometry, but it is not so clear that the design and sensitivity calculation capability in the CFD tools could get required geometric operations from a general standardized interface to a variety of solid modellers.

This geometry coupling is the foundation of an innovative approach to interactive turbine blade design using linearized Euler sensitivities developed in [29]. An Adaptive unstructured Euler equation solver produces a non-linear mean flow solution using the Bezier curve geometry as produced by the design tool. Additionally, the flow solver can compute linearized steady perturbation solutions for arbitrary geometry deformations of the blade shape. The perturbations specify the geometric sensitivity coefficients for each flow variable, such as $\partial P/\partial x_i$ for the pressure derivitive with respect to the i-th Bezier control point. Since the design tool and the flow solver both use the Bezier control point geometry to represent the turbine blade profile, it is possible for the flow solver to compute the geometric sensitivity for displacement of any control point. By providing this data to the design tool, as the designer modifies the blade by interactively moving Bezier control points, the blade surface Mach number distribution can be updated in real time using a locally linearized approximation. Given the baseline pressure ($P_0$) from the non-linear meanflow solution, the pressure $P$ after altering the i-th control point by a displacement $dx_i$ can be computed from

$$P = P_0 + \sum_{i=1}^{n} \frac{\partial P}{\partial x_i} dx_i$$

to within the locally linearized approximation. For blade changes beyond some range, the linear approximation error increases and the meanflow solution plus a new set of sensitivities must be computed. The same geometric sensitivites can also be used to accelerate blade design optimization using a generic engineering optimization package such [24,28].

# 7 RECOMMENDATIONS

The recommendations made here are intended to define the "ideal" CFD system. Many of the technologies either exist or are under active development. The CFD system should be modular so that if new technology becomes available an existing module can easily be replaced. The modules should be able to communicate with each other via either a common file format, or a particular format and a subroutine library to extract data from the file. There are five basic modules: Geometry definition, mesh generation, flow solution, adaption and post-processing.

## 7.1 Geometry Definition

The best approach here is to use a solid model from a CAD program to define the geometry of the component to be meshed, then for the reasons outlined in section 3 the grid generation program should read this model directly. To achieve this for a number of CAD packages a standard set of subroutine calls to interrogate a solid model will need to be defined, each CAD vendor will then need to provide an interface using these calls.

### 7.2 Mesh Generation

For the reasons stated in section 2 mesh generation should be based on unstructured technology. The following proposed algorithm is intended to be as automatic and as efficient as possible. The method begins by triangulating the surface of the body with the mesh density based on surface curvature. This removes the need for any additional input through for example a background mesh file. For efficiency the ability to generate stretched elements aligned with, for example leading edges would be useful. A viscous mesh can then be generated by inflating this surface triangulation. Finally the interior should be tetrahedralized with a Delaunay based algorithm where points are inserted based on mesh quality rather than a background mesh. The majority of components are in place for such an algorithm, the only real need is the removal of the background mesh.

### 7.3 Flow solver

To converge in a reasonable time any flow solver needs to employ a multigrid or implicit solution algorithm. An attractive and efficient way to generate the multigrid levels is to agglomerate from the initial mesh to get the coarser levels and adaptively refine through point insertion to generate the finer levels. The two technologies exist to do this but to the authors' knowledge they have not yet been combined.

The use of implicit algorithms on unstructured meshes is the subject of much research. As with their structured mesh counterparts, it is unclear which algorithm is the most robust and computationally efficient.

### 7.4 Refinement

Refinement is most efficiently done by point insertion: this will always be faster than remeshing. The ability to perform one-dimensional refinement described in section 4 is an essential part of any adaptive scheme. In addition this adaption procedure can be used to generate a one-dimensional multigrid scheme to accelerate the solver in viscous regions.

### 7.5 Post processing

There are a number of flexible post processors available. At the very least the software must be capable of handling different element types and any mix of them. Visual3 and FIELDVIEW are the only general purpose codes that to the authors' knowledge can handle mixed meshes, though others are migrating in that direction. To be of use to designers any post processor must be able to reduce the large three dimensional data sets to simple x-y plots of interest to an engineer, for example pressure around a wing section. 3D views of solutions using color contour plots, while very nice for publicity purposes, have little use to designers.

## 8 ACKNOWLEDGEMENTS

The authors would like to acknowledge the General Electric Company for permission to publish this paper. In addition the authors wish to acknowledge Dr. Kim Bey of NASA Langley Research Center for supplying the advancing front mesh generation program originally authored by J. Peraire, K Morgan and J. Peiro. Thanks are due to our colleagues Dr. M. E. Braaten and Dr. D. G. Holmes for many helpful discussions.

## 9 REFERENCES

1 Khune, C. M., Uenishi, K., Leon, R. M. and Abdol-Hamid, K. S. "CFD based 3D Aero Analysis System for High-Speed Mixer-Ejector Exhaust Nozzles" AIAA Paper 94-2941, 1994.

2 Holmes, D. G. and Snyder D. D., "The Generation of Unstructured Triangular Meshes Using Delaunay Triangulation," Numerical Grid Generation in Computational Fluid Mechanics '88, Pineridge Press, 1988.

3 Dannenhoffer, J. F., "Automatic Topology Generation from Block Structured Grids," VKI Lecture Series January 1994.

4 Dawes, W. N. "The Extension of a Solution Adaptive 3D Navier-Stokes Solver Towards Geometries of Arbitrary Complexity" ASME Paper 92-GT-363, 1992.

5 Shepperd, M. S. and Georges M. K., "Automatic Three Dimensional Mesh Generation by the Finite Octree Technique," International Journal for Numerical Methods in Engineering, 32:709-739, 1991.

6 Graichen, C. M., Hathaway, A. F., Finnegan, P. M., Kela, A. and Schroeder, W. J. "A 3-D Fully Automated Geometry Based Finite Element Meshing System" ASME Paper 89-WA/CIE-4, 1989.

7 Peraire, J., Morgan, K. and Peiro, J. "Unstructured Finite Element Mesh Generation and Adaptive Procedures for CFD", AGARD Specialists Meeting, Loen, Norway, 24-25 May 1989.

8 Löhner, R. "Adaptive Remeshing for Transient Problems With Moving Bodies", AIAA Paper 88-3737, 1988

9 Weatheril, N. P., "Adaptive Inviscid Flow Solution for Aerospace Geometries on Efficiently Generated Unstructured Tetrahedral Meshes," AIAA Paper 93-3390, 1993.

10 Blake, K. R., and Spragle, G. S. "Unstructured 3D Delaunay Mesh Generation Applied to Planes, Trains and Automobiles," AIAA Paper 93-0673, 1993.

11 Barth, T. "Aspects of Unstructured Grids and Finite-Volume Solvers for the Euler and Navier-Stokes Equations," AGARD report 787, March 1992

12 "Overview of PARASOLID, Version 5", EDS Unigraphics, Parasolid Business Unit, Parker's House, 46, Regent Street, Cambridge, UK, 1993.

13 Connell, S. D., Holmes, D. G. and Braaten, M. E., "Adaptive Unstructured 2D Navier-Stokes Solutions on Mixed Quadrilateral/Triangular Meshes" ASME Paper 93 GT-99, 1993

14 Connell, S. D. and Braaten, M. E., "Semi-Structured Mesh Generation for 3D Navier-Stokes Calculations" GE Research and Development Center, Report No. 94CRD154, August 1994.

15 Kallinderis, Y. and Ward, S., "Prismatic Grid Generation with an Efficient Algebraic Method for Aircraft Configurations," AIAA Paper 92-2721, 1992.

16 Melton, J. E., Pandya, S. A. and Steger, J. L., "3D Euler Flow Solutions Using Unstructured Cartesian and Prismatic Grids" AIAA Paper 93-0331, 1993.

17 Löhner, R., "Matching Semistructured and Unstructured Grids for Navier-Stokes Calculations," AIAA Paper 93-3348, 1993.

18 Prizadeh, S., "Unstructured Viscous Grid Generation by Advancing Layers Method," AIAA Paper 93-3453, 1993.

19 Connell, S. D. and Holmes, D. G., "A 3D Unstructured Adaptive Multigrid Scheme for the Euler Equations," AIAA Paper 93-3339, 1993.

20 Holmes, D. G. and Connell, S. D. "Unstructured, Adaptive, Finite Volume Solution Methods for Fluid Dynamics" 7th IMACS International Conference on Computer Methods for Partial Differential Equations (IMACS PDE-7), Rutgers University June 22-24 1992.

21 Mavripilis, D. J. "Three Dimensional Unstructured Multigrid for the Euler Equations", AIAA 10th Computational Fluid Dynamics Conference, AIAA Paper 91-1549-CP, 1991.

22 Braaten, M. E. and Connell, S. D. "A 3-D Unstructured Adaptive Multigrid Scheme for the Navier-Stokes Equations" GE Research & Development Center, Schenectady, NY, Report No. 94CRD146, August 1994.

23 Venkatakrishnan, V. and Mavripilis, D. "Agglomeration Multigrid for the 3D Euler Equations" AIAA Paper 94-0069, 1994.

24 Shelton, M. L., Gregory, B. A., Lamson, S. H., Moses, H.L., Doughty, R. L. and Kiss, T. "Optimization of a Transonic Turbine Airfoil Using Artificial Intelligence, CFD, and Cascade Testing", presented at ASME 1993 Gas Turbine Conference, Cinncinnati Ohio, June 1993.

25 Lamson, S. H. "Bezier Curves, Aerodynamics, and Optimization", presented at SIAM National Meeting, San Diego, July 26, 1994.

26 Korakianitis, T. and Papagiannidis, P. "Surface-Curvature-Distribution Effects on Turbine-Cascade Performance", Journal of Turbomachinery, Vol 115, 334-341, 1993.

27 Burgreen, G. W., Baysal, O. and Eleshaky, M. E. "Improving the Efficiency of Aerodynamic Shape Optimization", AIAA Journal, Vol 32 No. 1, 69-76, 1994.

28 Ashley, S. "ENGINEOUS Explores the Design Space", Mechanical Engineering, Feb. 1992, Vol. 114, Number 2, p. 49-52.

29 Homes, D. G. Private communication, 1993.

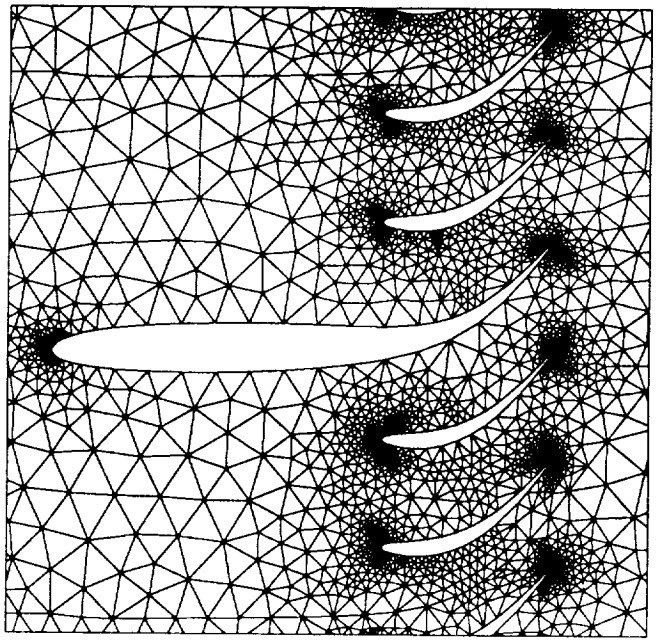


Figure 1 Example inviscid mesh

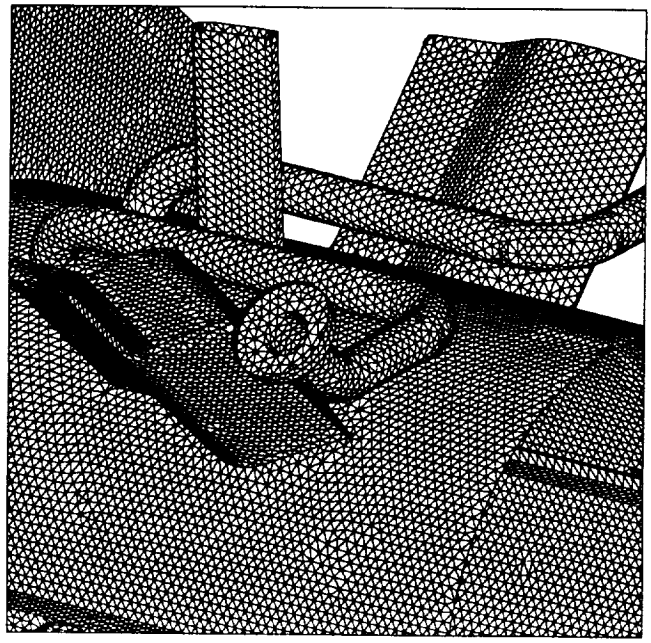
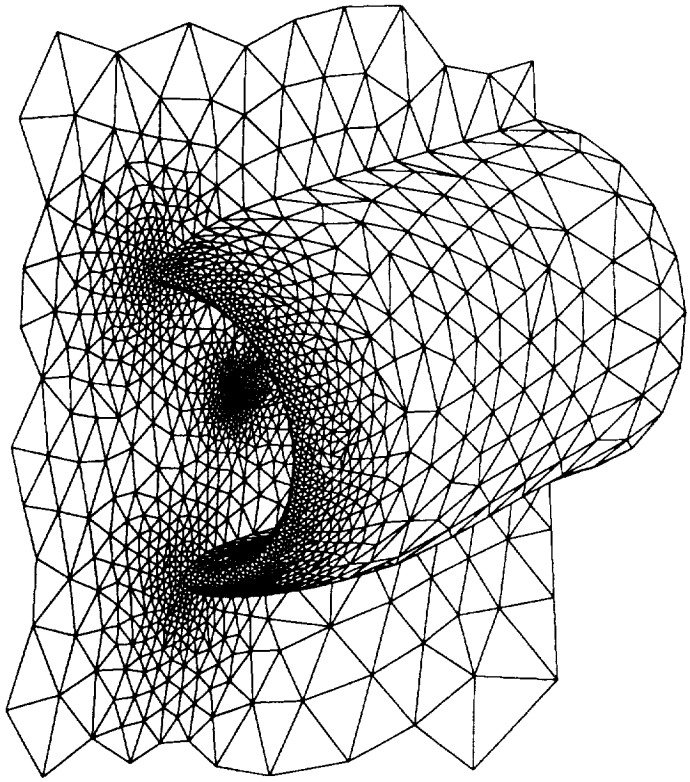
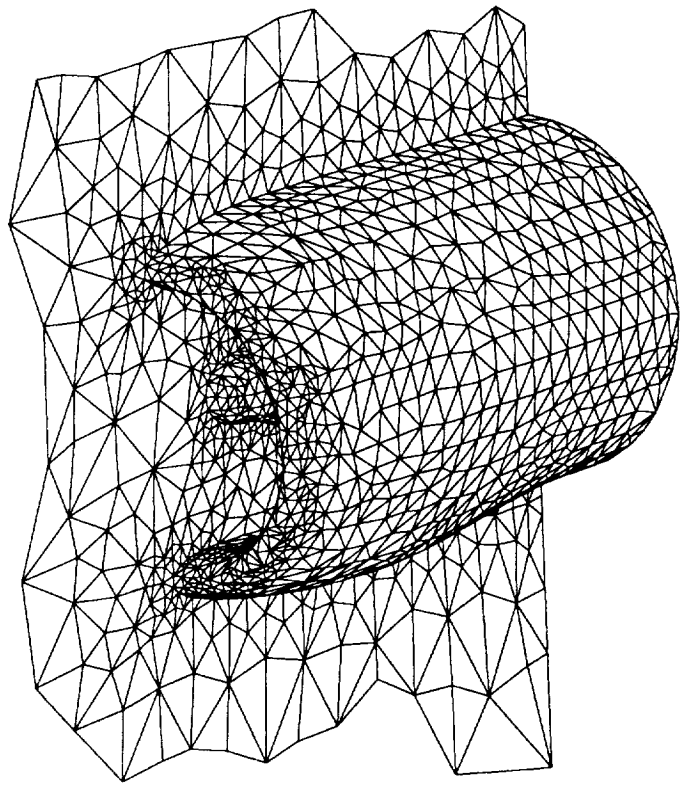


Figure 3 Surface mesh for 3D geometry



advancing front, nn=5805, nc4=28407



octree nn=5652, nc4=26175

Figure 2 Comparison of advancing front and octree meshes

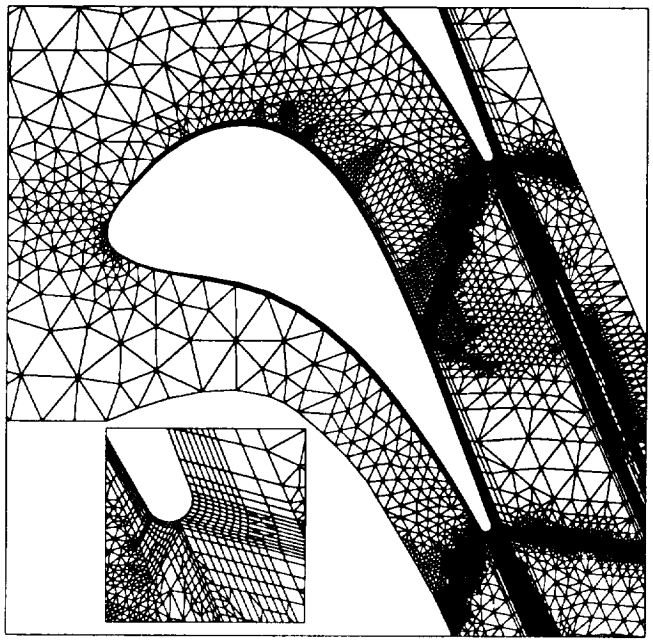


Figure 4 Mixed triangular/quadrilateral mesh

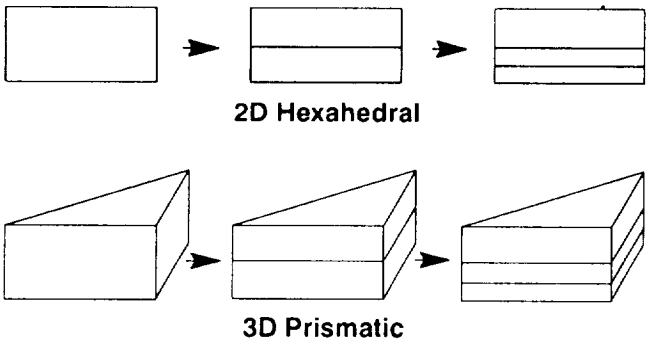



Figure 5 2D and 3D directional cell division

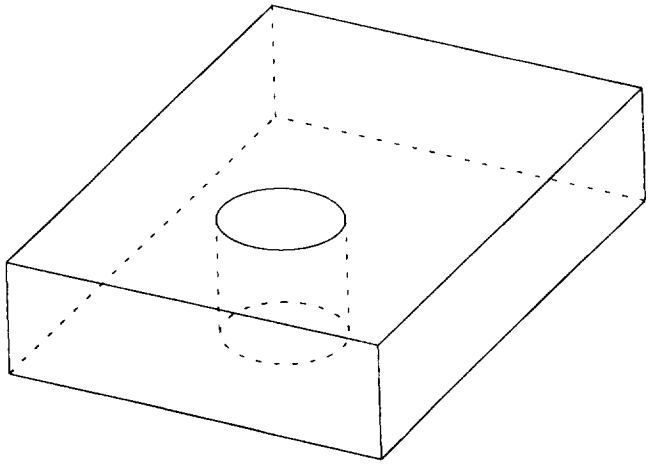


Figure 6 Cylindrical hole in box

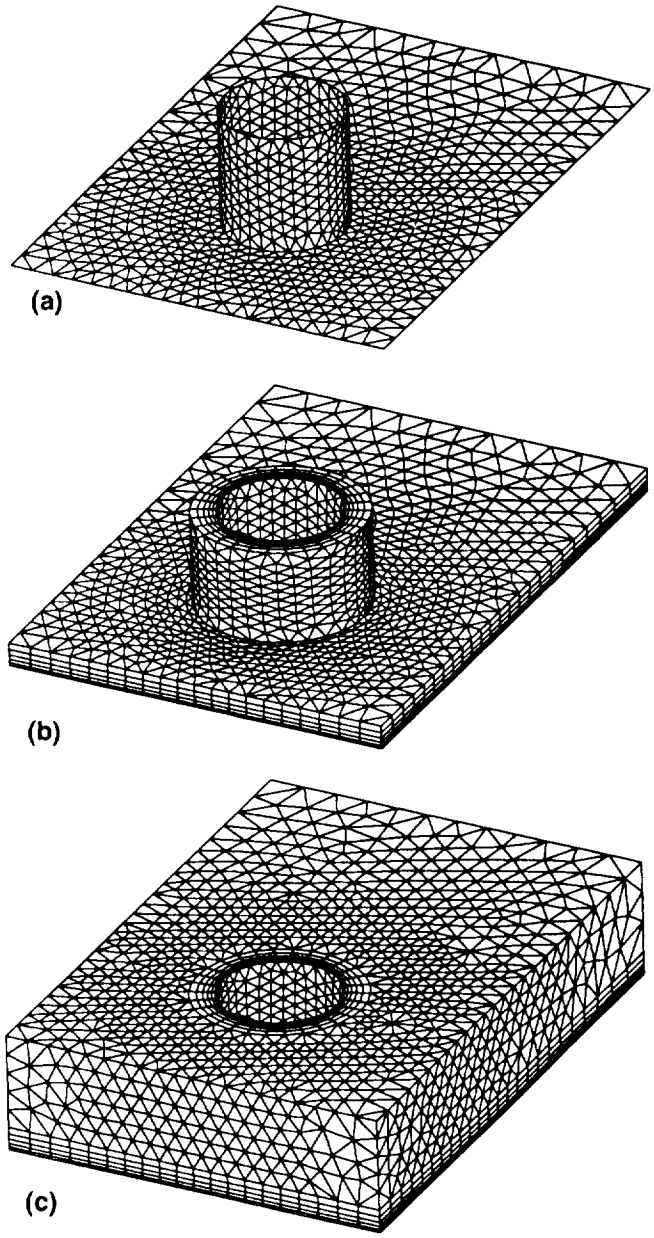



Figure 7 Stages of viscous mesh generation

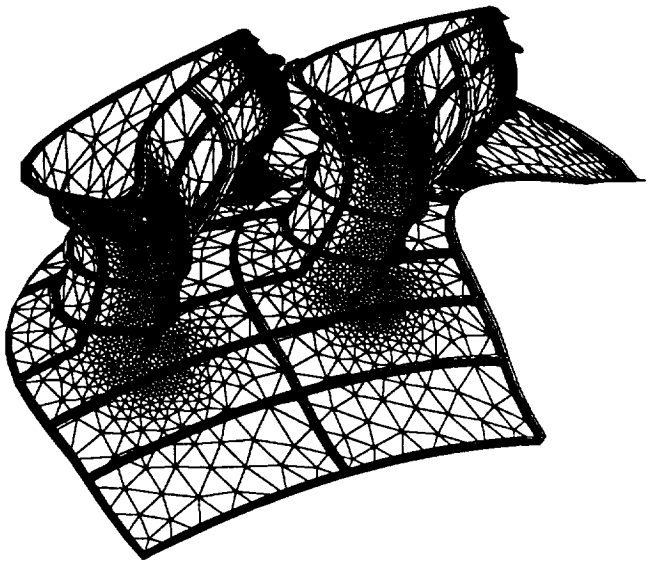


Figure 8 Prismatic mesh for turbine

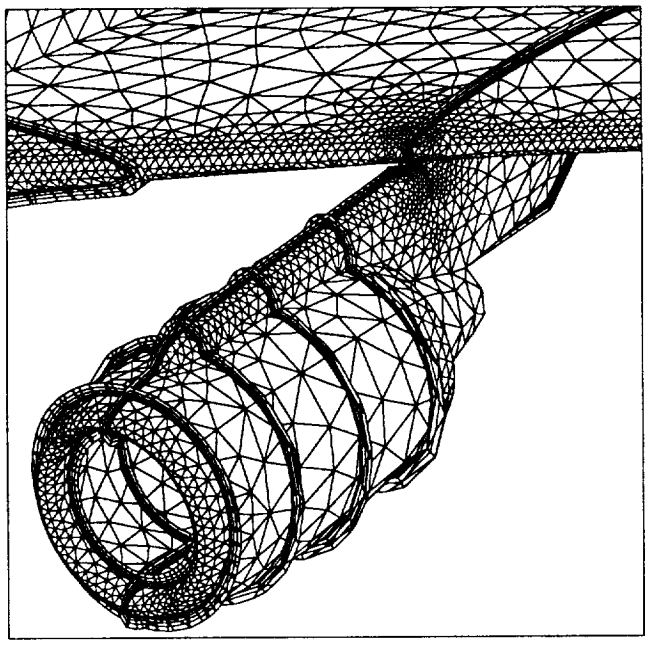


Figure 9 Prismatic mesh for installed nacelle

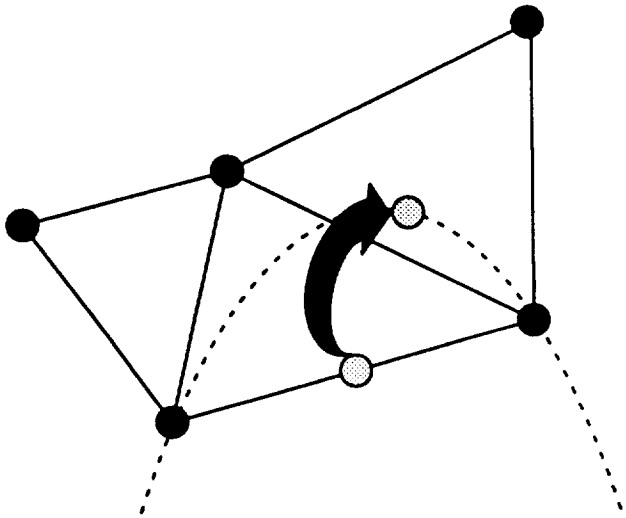


Figure 10 crossover problem for refined mesh

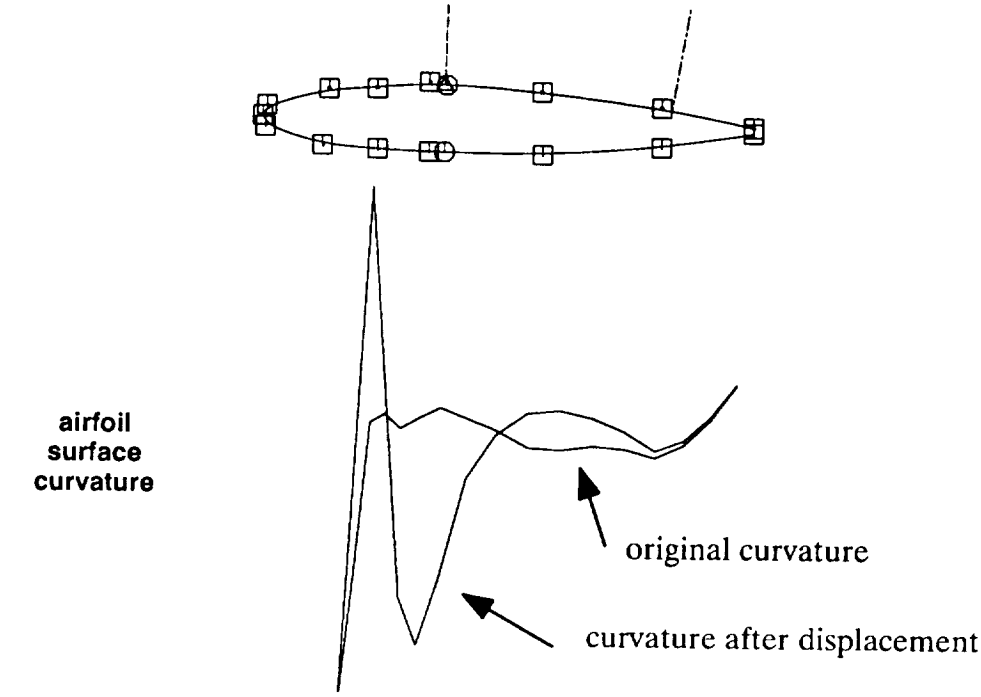



**Figure 11 Response of surface curvature to displacement of a single cubic spline control point on the upper airfoil surface.**

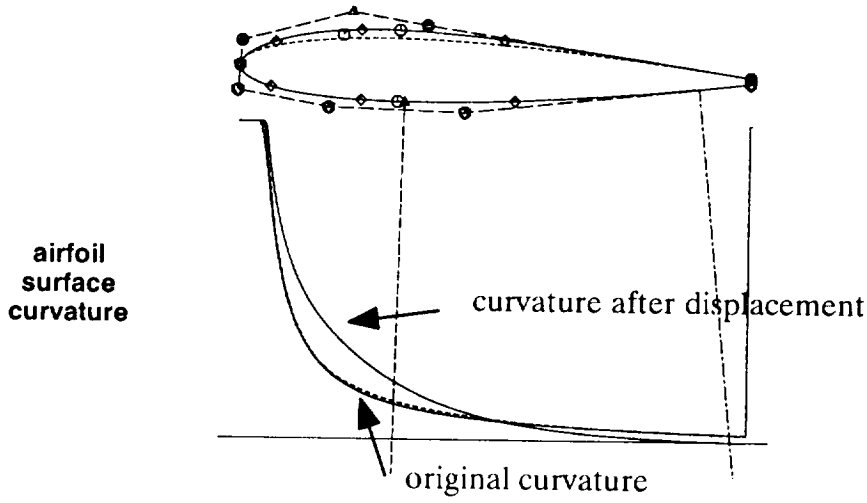



**Figure 12 Response of surface curvature to displacement of a single Bezier control point (Bezier curve of degree 4).**

19951223 06

N95-28727

# GRID GENERATION - A VIEW FROM THE TRENCHES

David Ives, Robert Miller, William Siddons, Kevin VanDyke[1]
Pratt & Whitney
East Hartford, CT 06118

## ABSTRACT

This paper presents "A view from the trenches" on CFD grid generation from a Pratt & Whitney perspective. We anticipate that other organizations have similar views. We focus on customer expectations and the consequent requirements. We enunciate a vision for grid generation, discuss issues that developers must recognize.

## BACKGROUND

The Engine Industry. Our grid generation goals are best understood in the context of the turbine engine industry, which is in the process of re-engineering with the premise that turbomachinery is a commodity in a mature business. This implies that specialists are used only in the core technologies, otherwise generalists will be supplemented by outsourcing of labor and/or purchase of vendor codes to meet business needs. The new paradigm of continual process improvement emphasizes driving costs down and reducing cycle time. The user of a CFD system is not a PhD level CFD specialist but rather a generalist designer. This represents a fundamental change from the last decade. The tight coupling of reduced design cycle time with time-to-market and bottom line profitability in this responsive business environment sets new CFD process requirements.

The Designer
A typical designer must balance theory, experiment, experience, and calculations covering a wide span of disciplines including CAD, aerodynamics, heat transfer, structures, and manufacturing. A designer is usually not a CFD expert. There is simply not time to develop in-depth skills in a specific area because of the wide range of skills and knowledge needed for design and because of the requirements to meet design deadlines.The designer may typically have a four month hiatus between sets of analyses, where he or she closely monitors component tests to achieve the necessary physical "feel" for the component. Thus the design systems must be easy to learn and relearn under tight schedules.

Flow Solver Success. Three dimensional viscous flow predictions have recently become validated in the engine industry as powerful design/analysis tools. Significant component improvements have been demonstrated for CFD-based designs. These recent successes are producing increasing pressure for expanding CFD capabilities to new component application areas. We have more opportunities now in CFD than we have ever had! Consequently, increasing numbers of component design systems are transitioning from correlation based algorithms to 3D viscous CFD analyses "tuned" with experimental test data. The grid generators must handle these new components.

Flow Solver Speed. A few years ago, the state of the art in industry was running jobs in queue on a supercomputer. Users often waited days for results for a single engine component. Then came a revolution: the use of networked workstations which currently produce overnight turnaround with high resolution. For example the flow analysis of a fan or compressor blade row, including details such as shrouds and tip clearances, is routinely done overnight. Network computing and more user friendly systems fueled a large expansion of the CFD user base to include the designers themselves, who

[1] Pratt & Whitney, West Palm Beach, FL



PRECEDING PAGE BLANK NOT FILMED

now run the overwhelming majority of CFD calculations at P&W. Flow calculations with one half million grid points are typical, with some cases employing 1.5 million grid points.

Grid Generation Turnaround. For simple components, such as a fan or compressor stage, specialized grid generators (developed over a ten year period) create the required grids without user intervention or even inspection. For complex new 3D components such as nacelle installations, complete combustor/diffusers, rocket turbopumps, etc., current grid generators may require days to months of elapsed time. In other words, the elapsed flow solver time can be a small fraction of the elapsed grid generator time!

Grid Generation Capability. Although current grid generators are rather general and applicable to a wide range of problems, some problems remain unresolved. For instance current grid generators currently cannot cope with some new applications such as the highly swept, highly wrapped (staggered), close-spaced periodic centrifugal inducer blade elements with multiple sets of embedded splitters as encountered in modern rocket turbopumps.

The Designer is the Customer. The designer ultimately decides which flow solver and grid generator are used, and which ones become "trashcan" CFD codes. Those that best meet the designer's needs get used. Those that do not best meet the designers' needs are discarded, even if such codes are "technically" superior. As a practical matter, the CFD grid generator developer must have a tight communication link with both the designer and the flow solver developer, so that the right needs are answered. The grid generator must integrate perfectly with the flow solver, and this requires constant communication as the target flow solvers and computing systems change regularly The key to this is understanding that the designer is the ultimate customer, and the flow solver/grid generator combination must meet design needs. This situation is like buying a house, where the advice is "location, location, location." Here the advice is "designer, designer, designer;" it is that simple. In practice, it is fully recognized that ,for logistical reasons, there may be "gatekeepers" between the developer or vendor and the designers. This does not alter the fact that the designer is the ultimate customer.

## VISION

CFD Process Vision. The requirement for the CFD process in this new market-driven business environment is **one day** turnaround for the everyday aerodynamic design CFD cycle illustrated in the first slide. This cycle is composed of inspection of the preceding nights' run, conceptualization of modifications expected to improve performance, defining the new geometry using a CAD system, CAD-to-grid geometry transfer, grid generation, flow boundary condition specification, flow solution, and post-processing. The flow solver unattended running occupies 16 hours of this cycle. This new design may have a different topology than previous cases. The grid generation occupies **one hour** of this cycle, leaving at least four hours for the designer to conceive and define a new geometry to improve on the current design. This 24 hour design-analysis cycle is in tune with the human circadian rhythm, allows a large number of design cycles, and cuts time to market for new products. Each of the CFD system components must be sufficiently automated to allow the designer to operate them on a sporadic basis with minimal specialized training and support from CFD "experts". The second slide illustrates that others also recognize this one day turnaround requirement.

Grid Generator Requirements. The requirement is reliable **one hour** grid generation turnaround for one-time geometries when run by designers. The system must incorporate CAD to grid links which resolve tolerance issues, and produce grids with quality good enough for the flow solver. The designer must feel that the grid generation process is under control and predictable, since the grid generation process is now viewed as part of an overall CFD process which is expected to produce

engineering answers in a day. The one hour grid turnaround requirement was derived by starting with the desired customer-defined goal and working backward.This requirements-driven approach will produce technologies significantly different (and more responsive to designer needs) than will a grid code written in isolation and then "thrown over the wall". We do not envision a new grid generator replacing our specialized grid generators (which usually run in at most a few minutes); in general the geometries we wish to analyze will be complex and significantly different from previous geometries. We also require that the grid generation be more automated with minimal user interaction to decrease variation between users, and to reduce training and retraining costs. Remember, every time the user must provide input, the variation in the grid increases! An automated approach for non-specialist customers is even more important for vendors than for internal developers, since local help and training may not be quite as close.

This **one hour** grid turnaround requirement is reality and must be fully accepted by CFD process providers. This is not an easy thing to do, given the current evolving state of grid generation. The development of grid generators to meet the one hour turnaround requirement may seem daunting taken out of context. Reliable one hour grid turnaround by generalist design users for up to a million cells is believed realistic based on hardware improvement projections and significant software process enablers discussed below. The one hour grid inside the one day CFD analysis will be the routine working environment of tomorrow!

Evolution or Revolution? Looking at how we have selected and deselected grid generation codes over the last decade at P&W, it becomes clear that we tend to hold on as long as possible to a technology that works, and only change codes or vendors when the current codes cannot effectively solve our problems in a timely manner. When we finally made the change to new codes, the technology always was a revolutionary jump, not an evolutionary improvement. For example, we switched from a command-line text driven grid generator to an interactive grid generator. As another example we ceased use of conformal mapping based generators when the coding changes required for flexibility became too time consuming because the basic code was not created with interaction in mind. When we switch codes, we typically evaluate the market leaders and "select" one within a single class of technology. Incremental improvements help postpone the date of transition to a new technology level, but do not change the outcome. We thus expect our next generation grid generators to be revolutionary, not evolutionary. Automation, which the dictionary defines essentially as "without human interaction," will increase and interaction will decrease.

The first generation of grid generators required typically thousands of command-line or script commands for complex cases. The second generation used graphical interaction to simplify the user interface, but still requires thousands of user decisions such as where to place a cursor and click. The interaction accelerated the user decisions and increased the probability of correctness, but did not significantly reduce the number of user decisions. Newer technology aims at reducing the number of decisions by automating tasks within the current process paradigm, such as the provision for more automated blocking or the ability to add components separately using templates. Simultaneously, altogether different gridding paradigms are emerging, such as the advancing front tetrahedral grid generators and the geometric grid generators discussed in Reference 3.

The interrelation between reliability and automation is illustrated in the fourth slide. A skilled user called on to make a moderate number of decisions will usually make the correct decisions, but as the number of user decisions grows, the probability of incorrect decisions grows. Given a **fixed** technology, the more user input "knobs" available, the higher the reliability if the user input is flawless. These trends interact to produce a reliability that at first increases with the number of user decisions, reflecting more user knobs to control special cases, but then decreases as the number of user

decisions grows further due to user errors. The idea is to automate the processes to move the left side of this curve dramatically upward and to the left. This movement, labeled "progress", is where the current frontier lies. Specialized grid generators tuned for limited geometries tend to have high reliability and few decisions. This is typical of the specialized cascade grid generators we have developed over the last 20 years. They are not expected to be displaced by new technology since they already fully meet the design needs for their limited scope of application.

Revolution Enablers Recent hardware, software, communications, and flow solver improvements have dramatically altered the computing and coding environment, making the one hour grid turnaround requirement a realistic goal.

Significant hardware advances have occurred in cost, speed, core storage, disk storage, and graphics. Workstations have lowered the cost per computational unit of work by order(s) of magnitude. Development has been accelerated by computing environments where there is no (or only a negligible) chargeback based on use. Developers can afford to experiment with new methods which show promise without having to justify the computing expenses. The impressive speed of modern workstations exceeds that of a single IBM 3090 processor of just 4 years ago. Just ten years ago, it was difficult to get authorization for 2 megabytes of core on a mainframe, now workstations typically have 32 times this amount. With this much core, one can try out new ideas quickly by not being concerned (in the 1st cut) with storage. The disk storage of typically 10 "cylinders" of ten years ago is now replaced by disk storage typically 100 to 200 times as much. This allows room for better documentation and stored verification cases, improving the code quality. The field of 3D graphics is about to explode. While there have been 3D graphics workstations available, the workstation market sales leader has announced that their next CPU chip will contain three subsystems: an integer unit, a floating point unit, and a graphics unit. You evidently will not be able to buy one without powerful 3D graphics capability! Others are developing increasingly inexpensive 3D graphics cards. A personal computer company has announced a 3D version of their "Quick Draw" operating system code which can display 3,000 facets in real time without an accelerator card. Since grids are 3D relationships in space, graphical visualization is an absolute necessity for the development of grid codes. This holds true even when automated grid codes do not need graphics for running; they still need the graphics for code debugging.

Significant software and communications advances include overlapping windows, the "make" facility, code checkers, sorting and searching algorithms, publications, and the Internet. The overlapping windows and concurrent processes capabilities allow development work to proceed on several different fronts at once. The UNIX operating system and the "make" facility simplify the construction of large codes. Typically a flow solver of ten years ago would range from 2,000 to 8,000 lines. Todays' flow solvers and grid generators often range from 50,000 to 150,000 lines of code. Progress on the code quality front has also proceeded on a rapid track. The **ftnchek** FORTRAN code checker of Reference 4 is capable of finding errors (such as uninitialized variables) in code even six levels deep in subroutines. The **lint** checker serves similar functions for "C" code. Computer science has contributed by establishing an "object attitude" relative to modular coding which again has allowed large, complex projects to be successfully completed. The knowledge of rapid sorting and searching algorithms has been developed and published extensively, and much faster codes can result.There are now a number of popular magazines and books devoted to coding techniques, while there were only a handful ten years ago. The Internet can be viewed as the world's biggest, best library as well as a source of technical news, technical papers, and assistance. It also simplifies collaboration between developers on a project.The power of this resource is growing at an exponential rate.These hardware and software advances work together to bootstrap the rate of code development; and this process will continue. We know of developers today who produce at a rate 2 to 8 times higher than

ten years ago.

Another enabler is the relative insensitivity of modern finite volume flow solvers to grid element quality (distortion) compared to the finite difference flow solvers of the past.The law of diminishing returns implies that once a grid is "good enough" for the flow solver, further improvements are hard to come by and might not be economically or time justified. This presumes that predictive grid quality measures for the flow solver are known, so that "good enough" is a measurable quantity. The simple grid quality measure "flatness" described in Reference 3 has worked well for our Euler flow solver. By relaxing the grid requirements from "perfect" to "good enough", new grid generation paradigms become feasible. Of course the grid must still be fine enough to resolve significant features so that grid independence is achieved.

## USER NEEDS

Needs. We asked developers, designers, and other CFD users at P&W to identify their grid generation needs, and were inundated. The results were merged, categorized, and are presented in the attached slides. They are a "wish" list based on our current experiences, and do not include all possible requirements.

Implications. An examination of the needs listed in the slides leads one to conclude that there are certainly a lot! This implies that the grid generator codes of the future will be large. These large codes will have more links between sections, greatly increasing the importance of coding quality standards and error avoidance procedures. Vendors may have an advantage with large codes since they can spread their costs over many customers, if there are only a few vendors. Another possible scenario is cooperative work between developers in different organizations to each address different needs, with coordination over the Internet. The National Grid Project and the NASA IGES effort are examples of such cooperation. An interesting topic for discussion is whether one dominant vendor will emerge, as in personal computer operating systems, or whether cooperative efforts of smaller groups will be the norm.

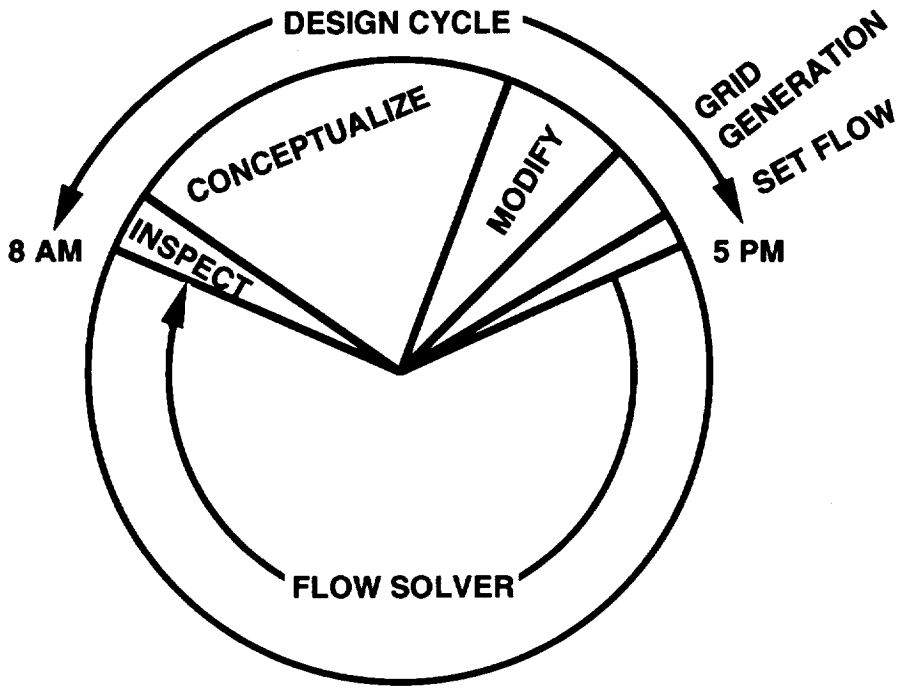
## CONCLUSIONS

This presentation concentrated on grid generation requirements from a customer-driven viewpoint. We first identified the designer as the customer, then discussed design requirements and presented perceived designer needs. We assert that the grid generation requirement is **one hour turnaround** for unique complex geometries on a workstation by a designer.We believe that this is an achievable goal, given significant enabling hardware and software developments. In closing, the bottom line is to recognize that the customer is the designer, and to be customer-driven.

## REFERENCES


1. Boerstoel, J.W., Spekreijse, S.P., and Vitagliano, P.L., "The Design of a System of Codes for Industrial Calculations of Flows around Aircraft and Other Complex Aircraft Configurations," 10th AIAA Applied Aerodynamics Conference, Paper # AIAA-92-2619-CP, Palo Alto, CA, June 1992.
2. Rubbert, P., "CFD and the Changing World of Airplane Design," AIAA Wright Brothers Lecture, Anaheim CA, Sept 1994.
3. Ives, D. C., "Geometric Grid Generation," Proceedings of Workshop on Surface Modeling, Grid Generation, and Related Issues in CFD Solutions," NASA CP#3291, NASA Lewis Research Center, May, 1995.
4. Monoit, R, "**ftnchek**," available by anonymous ftp from research.att.com (192.20.225.2) in the directory /netlib/fortran or by electronic mail by sending the message "send ftnchek from fortran" to the internet address: netlib@ornl.gov

# One Day Turnaround CFD




# One Day Turnaround CFD

***we are not alone***

**"In aerodynamic design work for commercial aircraft, CFD calculation results for aerodynamic analysis should be produced ... with short turnaround times (for large, complex calculations, 1 day, say)"**

Boerstoel, J.W., Spekreijse, S.P., and Vitagliano, P.L., "The Design of a System of Codes for Industrial Calculations of Flows around Aircraft and Other Complex Aircraft Configurations," 10th AIAA Applied Aerodynamics Conference, Paper # AIAA-92-2619-CP, Palo Alto, CA, June 1992.

**"The significance of this is that the total elapsed time for redesigning and arriving at a wing which meets all aerodynamic and manufacturability requirements can now be envisioned to approach the order of one day!"**

Rubbert, P., "CFD and the Changing World of Airplane Design," AIAA Wright Brothers Lecture, Anaheim CA, Sept 1994

# One Hour Grid Generation

**The requirement is for reliable one hour grid generation turnaround for one-time geometries when run by designers. The system must include CAD-to-grid links which resolve tolerance issues and produce grids with a quality good enough for the flow solver. The designer has to feel that the grid generation process is under control and is predictable.**

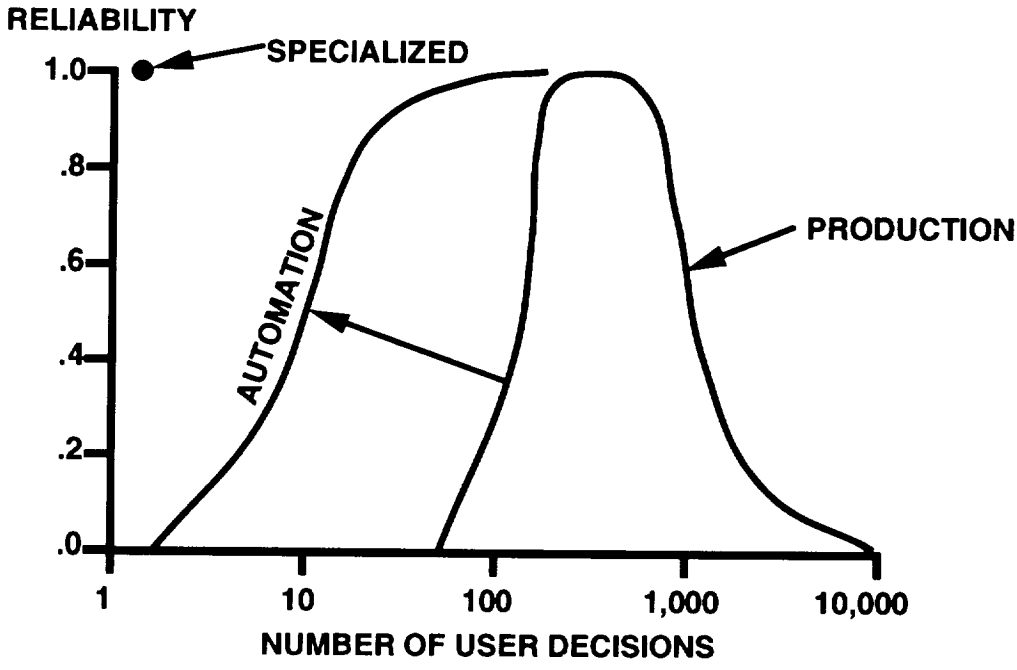
# The Designer is the Customer

*CFD only counts if it gets into the product*

**RECOGNIZE WHO THE CUSTOMER IS**
**RESPOND TO ACTUAL CUSTOMER NEEDS**
**FOR GRIDS, THE FLOW SOLVER IS ALSO A CUSTOMER**
**THIS REQUIRES LOTS OF COMMUNICATIONS**
**NO "OVER THE WALL" APPROACHES**

# DECISIONS

*A MEASURE OF AUTOMATION*



# Enablers

- Workstations
  - Speed/Cost
  - Large core
  - Large disk
  - Overlapping windows
- Make
- Code checkers
- Object attitude
- How-to publications
- Internet
- Fast 3D surface
- Graphics
  - In CPU:
    - integer
    - floating
    - graphics
- Robust Flow Solvers

# Process Reliability & Stability Needs

sufficient user control to ensure process nearly always works
soft landing after error with meaningful messages and waypoint capability
stability and reliability is usually more important than added features
check all user input for range & reasonableness, allow re-entry of data
reduce user input: every time you require user input, you introduce variation
be predictable enough for just-in-time grid generation
handle often imperfect CAD data with gaps, overlaps, tolerances
handle scales differing by 5 orders of magnitude
prevent bus errors and segmentation (memory) errors

# Ease of Use Needs

simplicity, lots of feedback on progress, active on-screen help
automate everything you can , then use graphical interaction for the rest
reduce need for grid generation knowledge & expertise
clear and consistent menus, messages, code appearance, process directions, and documentation
if something fails, suggest alternatives or corrective action on screen
provide visualization to track progress and detect failures
provide automated or simplified block decomposition
support general, non-restrictive script and data format with in-line comments
scripting and journal file for:

- batch running
- template construction by experts for "push-button" specialized uses
- coupling with other automated processes
- simplified re-learning and work organization for sporadic users
- restart from partially executed process waypoints
- reduced variation

# Integration Needs

block face points which are conceptually the same must be IDENTICAL
input and output source code to simplify integration into existing systems
read existing user formats, ie: VSAERO
batch running

# Standards Needs

write generic visualization files (ie: Wavefront .obj format)
accept standard CAD output formats (IGES, sterolithography, ...)
write Postscript files for design documentation
provide hardware platform independence where possible
all platforms should run identically
support the hardware platforms which designers have on their desks
use terminology and notation familiar to designers, where possible

# Developer Communication Needs

rapid response to critical bug reports, tell user if fix is not planned
good code/version control process and results tagging
Internet
- new features
- hints, suggestions
- feedback
- data base of outstanding and resolved bugs

# Miscellaneous Needs

common grid generator with international divisions and industrial partners
unstructured output capability, even for structured grids
adaptive tie-in to solver

# Implications

Large quantity of needs implies large codes needed

Large codes imply coding quality much more important

Will a dominant vendor emerge?
- ability to spread development costs
- like personal computer operating systems

Will cooperative efforts prevail?
- National Grid Project
- NASA IGES

# NASA OVERVIEWS

19951223O6

N95-28728

# APPLICATION OF CFD IN AERONAUTICS AT NASA AMES RESEARCH CENTER

Catherine M. Maksymiuk, Francis Y. Enomoto,
and William R. Van Dalsem
NASA Ames Research Center
Moffett Field, California

**Summary**

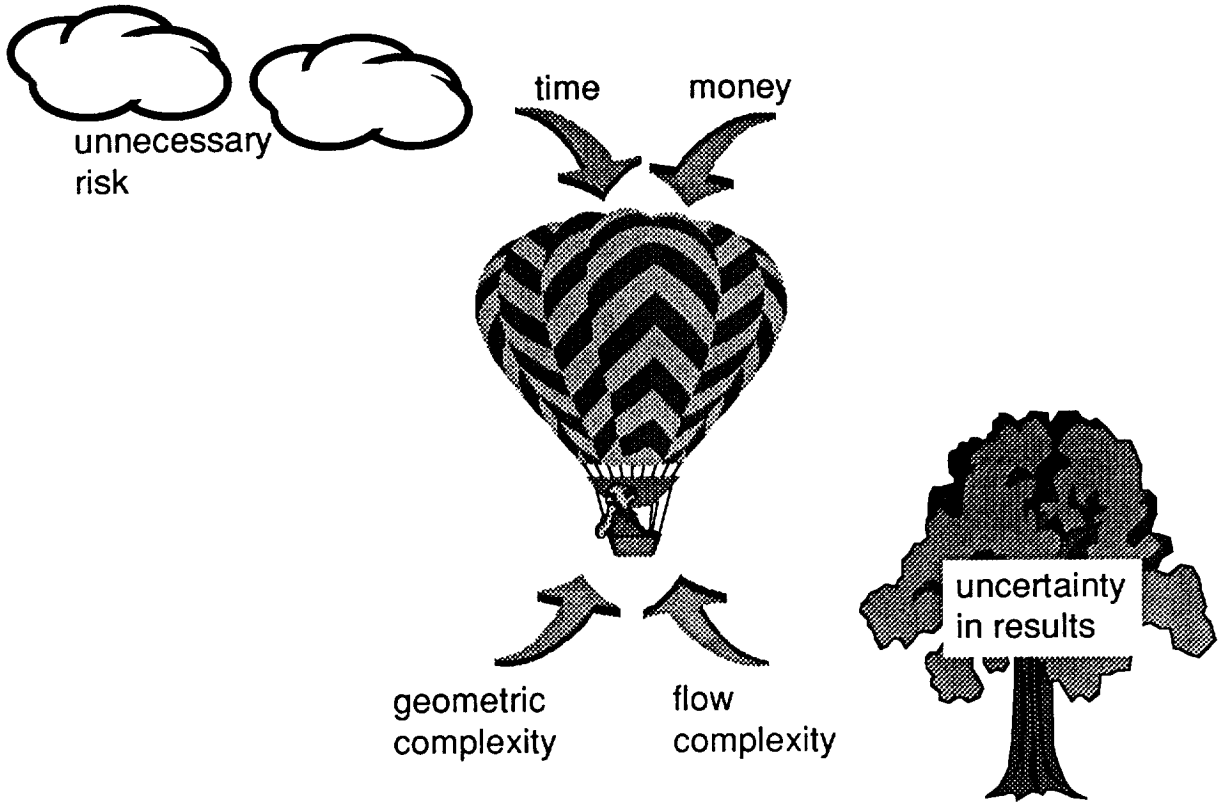
The role of Computational Fluid Dynamics (CFD) at Ames Research Center has expanded to address a broad range of aeronautical problems, including wind tunnel support, flight test support, design, and analysis. Balancing the requirements of each new problem against the available resources - software, hardware, time, and expertise - is critical to the effective use of CFD. Several case studies of recent applications highlight the depth of CFD capability at Ames, the tradeoffs involved in various approaches, and lessons learned in the use of CFD as an engineering tool.

## Outline

- **CFD as an Engineering Tool**
- **CFD Process in Action--4 Case Studies**
- **Assessment of Current Technology**
- **Conclusions**

PRECEDING PAGE BLANK NOT FILMED

## Balancing Requirements and Resources




When CFD is used as an engineering tool, care must be taken to achieve an appropriate balance between a project's modeling requirements and the associated cost. Time and cost constraints become direct limitations on how much complexity is retained in the surface model and how much approximation is made in the fluids model. If the models are oversimplified, the resulting "cheaper, faster" process might produce results that are not useful. Similarly, a simulation which models every imaginable detail might take so long that the results, however accurate, are too late to influence design decisions. The CFD engineer must keep his resources and requirements in equilibrium and must still avoid undue risk and uncertainty. This thought process should accompany every CFD application.

# Applied CFD at NASA-Ames

- Wind Tunnel Support
  - Model/Support/Tunnel Design
  - Wall/Sting Interference Prediction
  - Tunnel Data to Flight Extrapolation
  - Tunnel Data Analysis
- Flight Test Support
  - Flight Test Design
  - Flight Data Analysis
- Vehicle (or Component) Design & Analysis:
  - Concept Downselect & Preliminary Design
  - Design Optimization
  - Detailed Analysis

CFD is applied in many ways at Ames. These projects range from short-term, limited-scope analyses to detailed design studies or intensive analyses lasting many months. A corresponding range of tools and expertise are available to address these problems. The following slides will highlight some recent applications projects, showing how we rely on CFD and noting some continuing problems in the process.

# Advanced Subsonic Transport

M. Aftosmis, J. Melton

**Problem**

Investigate aft flow environment

**Constraints**

Turn-around ASAP

**Approach**

Vortex-dominated flow required high degree of geometric fidelity
Turn-around time precluded Navier-Stokes or any time-intensive grid system
Cartesian Euler with solution adaption offered quickest gridding

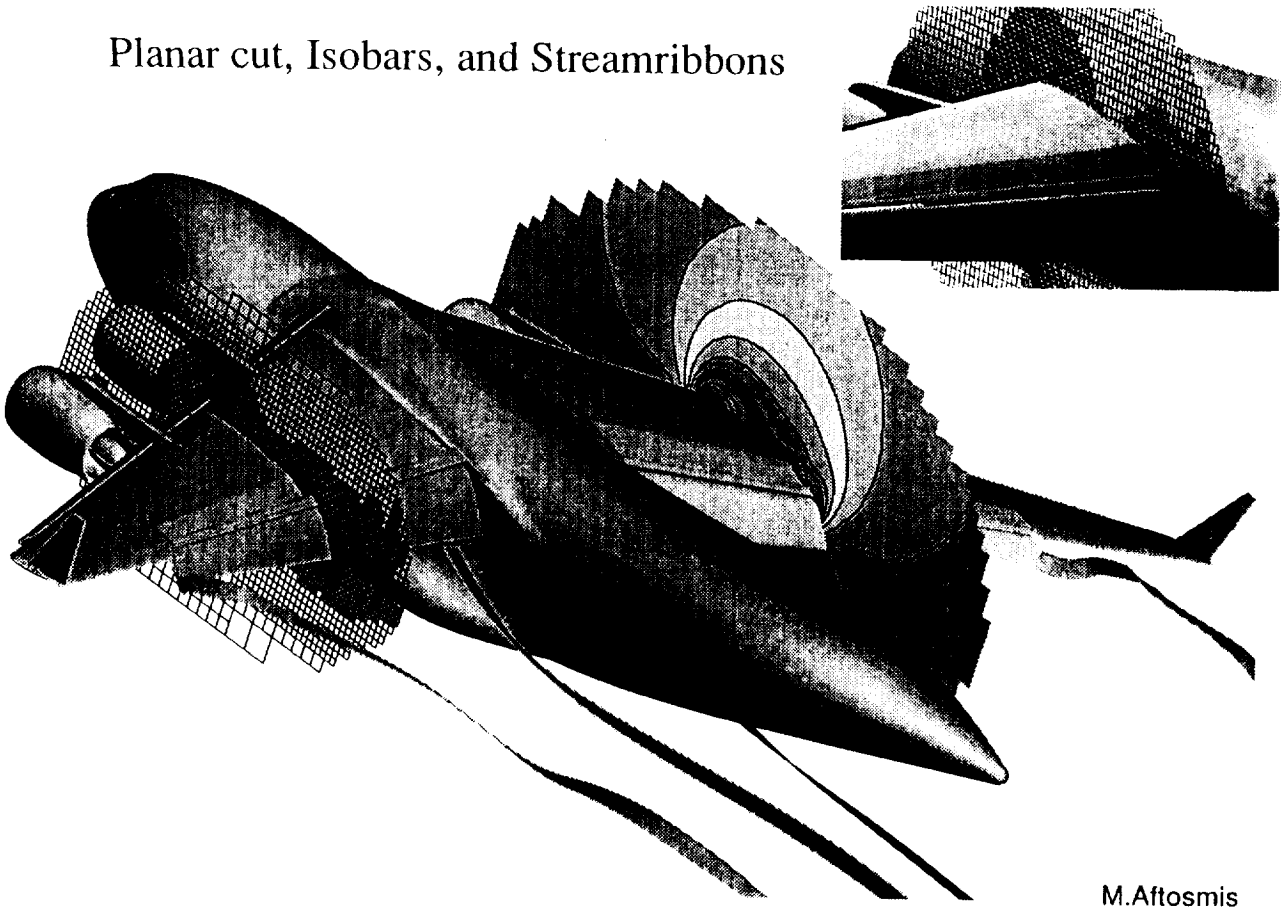
**Outcome**

Dominant flow structures were identified and results were coordinated with lower-order methods and water tunnel experiments.
Although the strength of the vortex structures is questionable due to the lack of viscosity, the trajectories are thought to be accurate.
The requesting agency, flight test center, and aircraft manufacturer were favorably impressed with the results and their timeliness.

**Lessons Learned**

CAD and surface grid generation are still a labor-intensive bottleneck.

Planar cut, Isobars, and Streamribbons



M.Aftosmis

# Navier-Stokes Analysis of Flow about a Flap Edge


## D. Mathias, K. Roth, J. Ross, S. Rogers, R. Cummings


**Problem**

Increased interest in aerodynamics and acoustics of high lift systems prompted an intensive analysis using CFD and wind tunnel tests

**Constraints**

Accurate representation of physics needed for full understanding of flow

**Approach**

Moderately complicated geometry was readily handled with algebraic and hyperbolic overset grids; a grid scripting system was created to facilitate handling future model with similar topology.

Flow conditions required the influence of viscosity but compressibility could be neglected.

**Outcome**

Initial results were used to locate pressure taps on model and proved to be an excellent prediction of wind tunnel data.

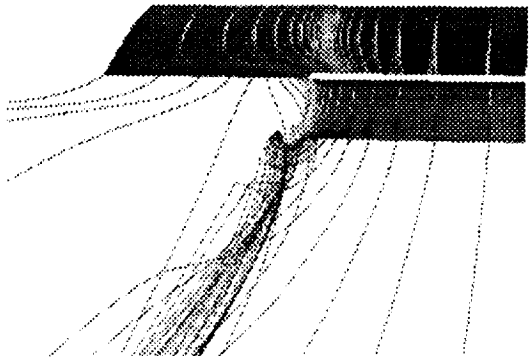
**Lessons Learned**

Engineering an overset grid system for a class of problems requires a significant investment up front, but can pay off in later similar problems.

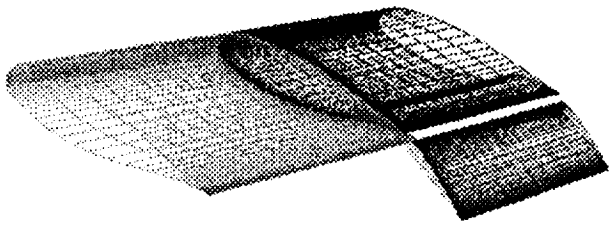
Proper location of overset grid boundaries is critical to obtaining desired convergence.

# Incompressible Navier-Stokes Analysis of a Half-Span Flap

**streamlines colored by static pressure**



**overset surface grids**



# Optimization of a Transonic Business Jet Wing

**J. Gallman, M. Madson, R. Kennelly, R. Chandrasekharan, V. Hawke, M. Hinson, D. Saunders, R. Mendoza, J. Reuther**

**Problem**

Design a new transonic business jet wing using numerical optimization and CFD

**Constraints**

Limitations on wing geometry due to manufacturing considerations
Performance requirements for $C_{D_{WAVE}}$, $C_{L_{MAX}}$ and no shock-induced separation
Short design period requiring efficient CFD and geometry modeling tools

**Approach**

Set of CFD codes included wing-alone full potential with numerical optimization, full configuration Cartesian full potential, wing-alone full potential with strip boundary layer, and wing-body structured Euler
Common geometry input from an automatic surface geometry program

**Outcome**

Strong influence of fuselage and nacelle on wing design
Design met most goals and was experimentally validated

**Lessons Learned**

Future aerodynamic design tools need to include fuselage and nacelle modeling, more accurate maximum lift estimates, and boundary layer effects

# Oblique All-Wing Supersonic Transport

R. Kennelly, D. Saunders, S. Cheung, C. Lee

**Problem**

Aerodynamic design of an Oblique All-Wing supersonic transport

**Constraints**

Thickness constraints due to passenger cabin requirements
Pitching and rolling moment constraints

**Approach**

Set of CFD codes included airfoil and wing full potential with numerical optimization, airfoil full potential with boundary layer, full configuration Cartesian full potential, and overset Navier-Stokes methods
NURBS representation used for design variables in airfoil aerodynamic optimization
Chord and thickness tapers designed with numerical optimization

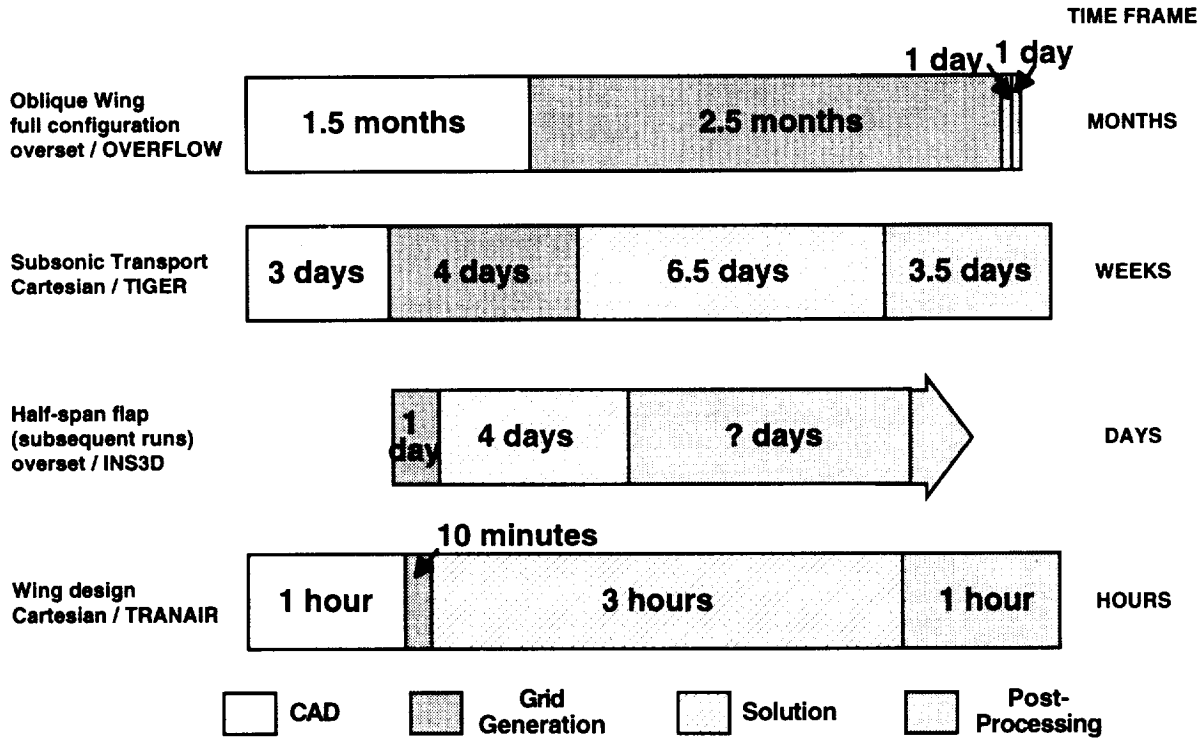
**Outcome**

Extensive geometric parameter studies led to an asymmetric planform design
Design met goals and was experimentally validated

**Lessons Learned**

Parametric surface definition was useful for making numerous design iterations
Full configuration analyses were indispensable in design process, but hampered by CAD to surface grid bottleneck
Navier-Stokes analyses took months

# Process Time for Case Studies



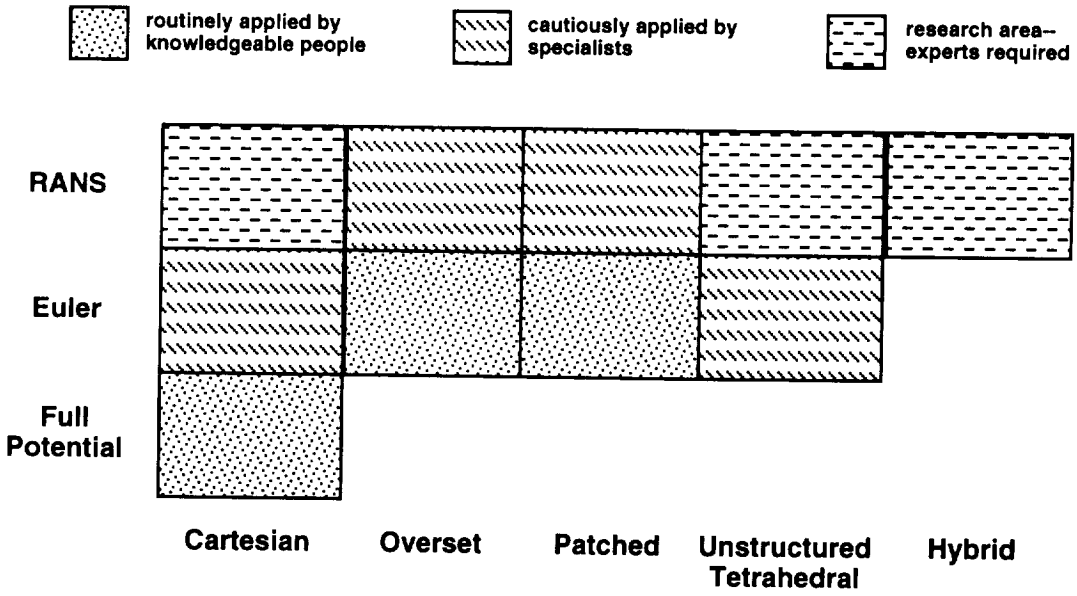

It is instructive to compare process times for the various methods and applications. The design cycles for the Oblique Wing and the Business Jet Wing took 1 year and 4 months respectively; the full processes are not represented here. The Navier-Stokes analysis of the Oblique Wing took several months. The full configuration was a complicated CAD model, and the grid required considerable effort. Once a complete overset grid system was obtained, changing the configuration by moving or removing any of the modular parts was relatively simple, and several such configurations were studied. Post-processing was automated to extract desired information from the solution files.

The full-configuration transport analysis is an example of a short-term, limited-scope project. The short process time was a major goal, which dictated the choice of method. The elapsed time for CAD modeling is a little deceptive, since several people worked on the model.

The Half-span Flap analysis is similar to the Oblique Wing design study. Considerable time was invested to set up a system that could be used for future work on similar configurations; such studies can now be run in a few days. Because the resulting database is used for continuing study, especially in connection with wind tunnel data, the post-processing time is shown as indeterminate.

The most efficient process shown is a TRANAIR run, concluded in only a few hours. The CAD process was actually a surface modeling process not involving conventional CAD software. Analytical surface modeling was done specifically to avoid having to work with a CAD model.

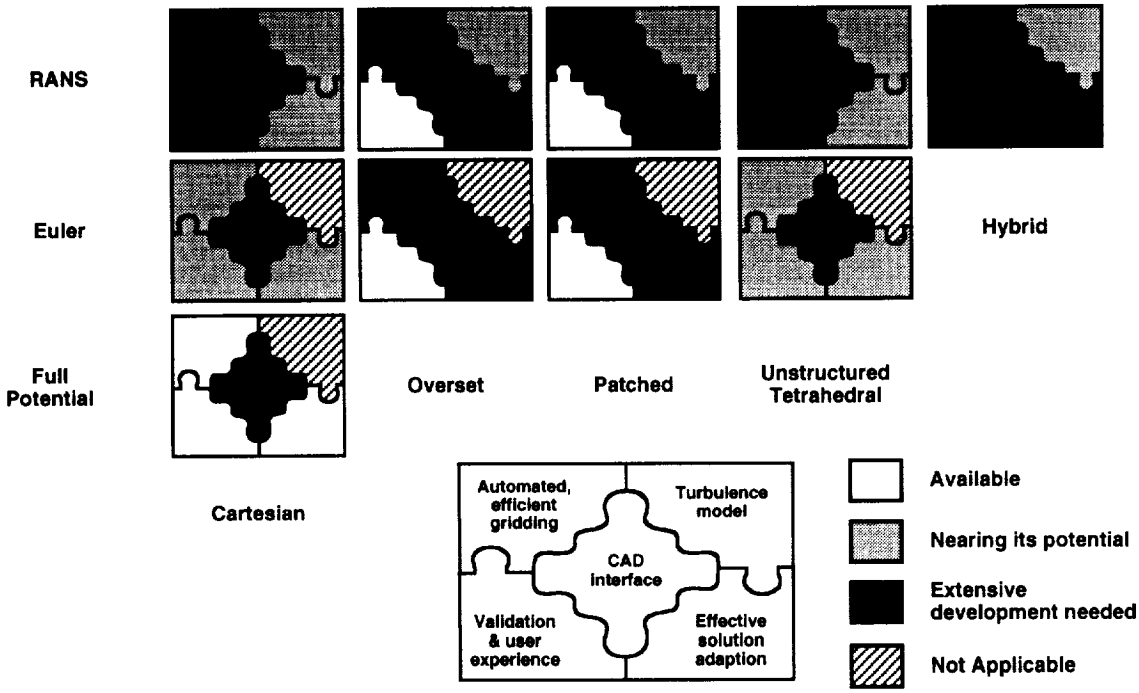
# Assessment of Current Technology




CFD software has traditionally been used by very specialized engineers and researchers. Lately, there has been some effort to make codes usable by people less experienced in CFD. Although progress has been made, most applications are still carried out by CFD specialists.

The most usable codes are those which have been available for a long time and have seen the most development or those which have achieved the most automation through extensive development or inherent capability. Experience with the grid generation procedure and the flow solver's limitations is still necessary for effective results. Less well-developed or well-validated techniques require more effort in grid generation, more understanding of the modeling in the flow solver, and/or more acceptance of uncertainty in the results. Several important methods still require considerable research before they can be routinely applied.

# Missing Pieces




No CFD method for complex configurations has a simple process path at this time. Despite the advances in standardization by NASA-IGES, the CAD/surface grid interface is still very cumbersome. Overset and patched methods lack automated gridding and are not conducive to efficient solution adaption. Navier-Stokes methods need attention in many areas; hybrid methods may be able to overcome the inherent inefficiency of unstructured Cartesian or tetrahedral grids, yet offer more automation of grid generation than overset or patched methods.

# Conclusions

- No one method is the best for all problems. Select the strategy which meets the deadline with the available resources and addresses the important physics of the problem.
- To address a wide range of problems effectively, a full range of tools must be available.
- There is a long process-time gap between Euler and Navier-Stokes analysis.
  - Unstructured methods have done well for Euler problems but have not been satisfactorily demonstrated for viscous 3D problems.
  - Structured methods for a class of problems can be automated for quick turnaround; generalized fast structured capability still does not exist.
  - Hybrid methods could be the answer, if they combine the best of each method.
- Fast Euler methods are attractive from a process-time standpoint, but users must be aware of approximate physics.
- For complicated or detailed geometry, CAD-to-grid is still a major bottleneck.

[redacted]
19951223308

N95-28729

# GEOMETRY LABORATORY (GEOLAB) SURFACE MODELING AND GRID GENERATION TECHNOLOGY AND SERVICES

Patricia A. Kerr, Robert E. Smith,
and Mary-Anne K. Posenau
NASA Langley Research Center
Hampton, Virginia

## Abstract

The facilities and services of the GEOmetry LABoratory (GEOLAB) at the NASA Langley Research Center are described. Included in this description are the laboratory functions, the surface modeling and grid generation technologies used in the laboratory, and examples of the tasks performed in the laboratory.

## 1 Introduction

Discussion of the concept of GEOLAB began in the late 1980s when computational fluid dynamics had demonstrated the capability to produce accurate and relatively inexpensive flow fields about complex geometries. The bottleneck for obtaining solutions, however, was the amount of time required to prepare the surface geometry and grid data for the flow field solvers. Typically, several months of intense activity was required to produce geometry data after which the first flow field solution could be obtained in a matter of hours. A similar amount of time was required each time a new configuration was undertaken. Computer-Aided Design (CAD) software and emerging grid-generation software potentially could increase the surface and grid data productivity, but it had to be organized and integrated into the overall computational environment. The concept of GEOLAB was to centralize those geometry activities requiring specialized (but broadly applicable over many disciplines and organizations) talents, software, and hardware and to decentralize those tasks which required individual engineering tuning. Training and communications were, and still are, the essential ingredients for the concept.

GEOLAB was initially formed in 1990 to support Langley researchers performing Computational Fluid Dynamics (CFD) analyses. GEOLAB focuses on high performance workstation hardware, CAD and grid generation software, and the skills to efficiently produce geometry data. The main goals of GEOLAB are to improve the efficiency in performing geometry related functions such as surface modeling and grid generation, to promote technology development of new geometry techniques, and the transfer of such development to customers both internal and external to the Langley Research Center. In the past four years, GEOLAB has expanded its capabilities in order to support projects involving conceptual design, model production, structural analyses, materials, multi-disciplinary optimization, and

PRECEDING PAGE BLANK NOT FILMED

reverse engineering.

Since its inception GEOLAB has reduced the time to perform a surface modeling and grid generation task defining a volume grid of a full aircraft complete with tail surfaces, fillets, pylons and engine nacelles from 6 months to 2 weeks. In addition, this also represents an approximate "order of magnitude" decrease in the cost of grid generation for a detailed Navier Stokes CFD analysis in a typical project.

# 2 GEOLAB Resources

GEOLAB is equipped with state-of-the-art hardware and software. The hardware includes 10 high-end Silicon Graphics workstations and 7 NCD X-terminals. The workstation consoles are located in a communal area near desk space where the X-terminals are located. A CYBERWARE laser digitizer and a Calcomp Flatbed Digitizer are also available in GEOLAB to enable mathematical surface reconstruction from physical objects and blueprints. The equipment is used primarily for the production of surface models, computational grids, and for software development; users must be involved in surface modeling or grid generation projects to qualify for accounts and use the equipment during prime hours. In addition, researchers working on parallel algorithms are given accounts in order to test their algorithms in a clustered workstation environment during non-prime hours. There are over 75 users including the GEOLAB team.

The GEOLAB software includes CAD, grid generation, and visualization tools that have been developed or acquired to facilitate the generation and analysis of surface representations, surface grids, and volume grids for both structured and unstructured analysis techniques. Among the tools currently in use in GEOLAB are: GRIDGEN, ICEMCFD, VGRID, GridTool, CSCMDO, SurfACe, SCAFFOLD, and VOLUME. Each of these software tools is briefly summarized in Appendix A.

A key element to productivity and efficiency in GEOLAB is the integrated computing environment provided by the system administration techniques employed. All user files and software files are cross-mounted so that all utilities are available on all machines. Network licenses are purchased when buying commercial software so that no workstation needs to be reserved for a particular function. Passwords are common across the machines. Access to general-purpose tools and utilities such as printers, manual pages, electronic mail, editors, etc. are available from any machine. An attempt is made to keep the workstations at the same revision of the operating system and to keep the amount of memory and CPU type uniform across the cluster. In general, the user is presented with the same environment and resources transparently, no matter which workstation is used.

The hardware and software are matched with a team of skilled specialists consisting of 5 civil servants, a National Research Council fellow, a university post doctorate, two university

civil servants, a National Research Council fellow, a university post doctorate, two university doctoral candidates, and 7 non-personal services (NPS) contractors. The members of the team have mechanical or aerospace engineering, computer science, and mathematics backgrounds. The contractor team members are responsible for performing most of the surface modeling and grid generation tasks in response to the Research and Technology Group at Langley and other external organizations. Also they are heavily involved in the development of specialized software tools that are needed to improve the efficiency of the process or to provide a custom tool to support a research project initiated in the Research and Technology Group at Langley. The civil service personnel are involved in the development of new methodologies for surface modeling and grid generation, data transfer and computing environment issues, and the administration of GEOLAB. While the expertise of team members lies in different specialty areas, most are cross-trained in all the software tools used in the laboratory in order to maintain a cohesive and flexible response to customers.

GEOLAB is currently located in the Scientific Applications Branch of the Information Systems Division, Internal Operations Group at the NASA Langley Research Center. The GEOLAB team has access to advanced tools for scientific data visualization, data base management, image processing, and high performance computing as a part of the Scientific Applications Branch. In addition, the team has ready access to other services within the Internal Operations Group that support wind tunnel model production and experimental testing. The availability of these capabilities greatly enhance both the scope of problems GEOLAB is able to address and the quality of the resulting work performed in GEOLAB.

# 3 GEOLAB Functions and Experiences

GEOLAB functions can be divided into 4 categories: (1) production surface modeling and grid generation tasks; (2) software development supporting surface modeling and grid generation; (3) research into new technology; and (4) training and technology transfer.

## 3.1 Surface Modeling and Grid Generation Tasks

GEOLAB performs a wide variety of production surface modeling and grid generation tasks enabling researchers to effectively proceed with their computational analyses. The details of these tasks include data repair such as adding missing features; blending features together; and reconstructing surface models from grid points, blueprints, or measurement of existing physical models. Other tasks include preparation of surface and grid data for parametric studies and production of denser or sparser grid sets.

Most production surface modeling and grid generation tasks take 2 to 6 weeks to complete depending on the complexity of the configuration, the accuracy required to support the intended analysis tool, the nature and condition of the input data, and the format of the

output data. Projects in which the input data is not in electronic format, i.e., coefficients or points in a design book, blueprints, or models for which a high degree of accuracy is needed to support Navier-Stokes analysis, may take 6 to 12 months to complete. Usually these projects are only undertaken if several groups of researchers have need for the data. GEOLAB also provides support for much smaller tasks such as data conversion, geometry visualization, and quick data verification for researchers. These tasks usually require anywhere from an hour to 3 days. Tool development tasks require anywhere from 2 to 12 months development time. The tasks are usually broken down into sub-tasks with specific deliverables and schedule. The researcher is involved at an early stage as the requirements are decided upon and as each major subsection is developed and demonstrated. This involvement also provides several points at which priorities may be re-assessed and adjusted.

## 3.2 Software Development

The software used in GEOLAB is licensed from commercial companies, obtained free from government sponsored development projects, or created in-house. The choice of which source to use is based on availability, cost and timeliness. CAD software, as a rule, is always licensed. Grid generation software is usually acquired from freeware sources or created in-house. Special purpose software is almost always created in-house. The programs SurfACe, GridTool, SCAFFOLD, and CSCMDO described in Appendix A are representative of the software developed in GEOLAB. The rationale for creating them in-house follows.

SurfACe (Fig. 1) is a software package used to evaluate the quality of surface grids. It allows visualization of the surface of multi-component configurations and the characterization of several grid quality functions on the surface. Since SurfACe is tailored to evaluate surface grid quality, it has been designed to expedite these operations. This gives SurfACe its primary advantage over systems such as FAST, a freeware visualization system or FIELDVIEW, a commercial visualization system.

GridTool (Fig. 2) is a surface grid generation program originally written to prepare unstructured surface grids for the VGRID unstructured volume grid generation system. GridTool has evolved to be a versatile utility bridging the gap between CAD geometry and grid generation programs. GridTool creates grids on patches that are projected to the CAD description, thereby easily generating grids that are on the precise CAD description of the surface.

SCAFFOLD (Fig. 3) is an interactive program written to recreate numerical geometry models from measured data obtained by scanning physical models with the CYBERWARE laser digitizer. SCAFFOLD provides the bridge between measured models and CAD models. It is an example of the specific software developed in GEOLAB to meet a specialized need across many organizations.

CSCMDO (Coordinate and Sensitivity Calculator for Multidisciplinary Design) (Fig. 4)

is an automatic volume grid generator that operates in a batch environment in conjunction with a shape optimization process. It is based on being given a surface grid for a high-speed civil transport configuration and an initial volume grid possibly generated with software such as GRIDGEN or ICEMCFD. CSCMDO automatically adjusts the volume grid to small changes in the shape of the configuration surface. Large changes in the surface shape, topology or number of surface grid points require a new volume grid from another source. CSCMDO was developed at the request of the Multidisciplinary Design and Optimization Branch of the Langley Research and Technology Group.

### 3.3 New Techniques for Surface Modeling and Grid Generation

The establishment of GEOLAB was based on many prior years of grid generation research in the Analysis and Computation Division at the Langley Research Center (LaRC). The development of new techniques is now an integral part of GEOLAB. There are two in-house project under way: (1) The rapid generation of airplane configurations using engineering parameters, and (2) The development of high aspect ratio unstructured grid generation techniques. Appendix B presents a short description of each of these projects.

## 4 Training and Technology Transfer

Besides surface modeling and grid generation tasks, GEOLAB also provides training in the software tools used in the lab. The tools may be commercial software or they may have been developed under contract for NASA to researchers at LaRC, other NASA centers, government agencies or industrial partners. Included in the list of tools supported to this extent are: GRIDGEN, ICEMCFD, GridTool, and CSCMDO. GEOLAB presents two grid generation classes a year on elements of grid generation, the use of GRIDGEN for structured grids, and the use of VGRID for unstructured grids. These classes include hands-on tutorials. During these sessions GEOLAB is dedicated to the training class to allow the participants access to the hardware. The lab environment is ideal for this type of activity with 10 machines in one area and easy access for the instructor. One-on-one training is provided on request in the use of specialized tools like SurfACe, GridTool, and CSCMDO. Over 200 people have been trained by GEOLAB since its inception.

GEOLAB is committed to technology transfer to the Langley researchers, other government and university centers, and U. S. industry. In addition to offering training, GEOLAB makes available its non-proprietary software tools upon request. By far, the most requested software is GridTool. At this writing, 80 copies have been distributed throughout the United States, in addition to its use at Langley.

Cooperative Agreements between GEOLAB and outside organizations can be made to produce surface models and grids. GEOLAB staff will demonstrate the software tools and

methodologies on geometry problems of interest to the requester. Agreements of this nature have been entered into with David Taylor Research Labs, Boeing Company, McDonnell Douglas Corporation, and Ford Motor Company. These agreements are normally coordinated through the Technology Applications Group (TAG) at Langley

# 5 Data Management and Communications

Data Management and Communications are an extremely important aspects of GEOLAB. Surface geometry is normally communicated in Initial Graphics Exchange Specification (IGES) or NASA-IGES (NIGES) files. This standard formatting of surface data allows rapid interaction with customers, both government and industrial, for surface geometry problems. Langley Wing Geometry Standard (LaWGS) and PLOT3D files are also used.

A data management system called PRISM is used to manage large quantities of data pertaining to a given project. PRISM arose out of the High Speed Airplane Integrated Research (HiSAIR) study and is comprised of a Graphical User Interface (GUI) and a software interface to a Database Management System (DBMS). The PRISM GUI uses the X Windows System and the Motif toolkit to provide a window based "point-and-Click" interface. PRISM used the commercially available Informix DBMS with a Sun SPARCstation currently functioning as the database server. The PRISM software provides for automatic searching or querying for particular data sets (or subsets) to the user's workstation, storing new data sets, and an administrative/security mechanism controlling access to shared databases.

# 6 Future Directions

There are several problem areas that GEOLAB would like to address in the future. The most important is the length of time it currently takes to produce a surface representation and computational grid for analysis. As was stated earlier, it usually takes 2-6 weeks to prepare the geometry and volume grid for a detailed Navier-Stokes analysis. To meet the needs of industry, this time must be reduced to hours. To meet this need, better tools are needed to run in a more automatic fashion with little or no human intervention. In addition, these tools must also be flexible and robust enough to handle computations dealing with a high degree of complexity.

Further work is indicated in other specific areas in which GEOLAB is currently involved. More diagnostic tools are required to guarantee that any given computational grid is satisfactory, thereby reducing the number of revisions needed to allow accurate modeling of the physical processes being studied. These tools need to be developed for computational structural mechanics problems as well as for computational fluid dynamics problems. Additional work is needed in the area of unstructured grids in order to allow the generation of un-

structured grids to handle viscous flows. Further work in the area of parameterizing surfaces with respect to design variables is needed to shorten the design and preliminary design cycles.

In addition, expansion of effort in other areas is indicated. New advances in the area of reverse engineering are to be studied. New techniques to relate 2-D CT scans to an accurate 3-D surface representation of a model or structural element are needed. Also, more robust algorithms for surface reconstruction from point clouds measured from laser digitizers are needed. More experience is needed within the GEOLAB team to become more familiar with grids for structural analysis.

As the team works to solve individual task requests, new tools are being generated and existing tools are being improved to increase the efficiency of all aspects of the grid generation process.

# 7 Conclusions

GEOLAB has become the focal point for geometry issues at Langley. It is providing an essential function for all continuum simulation in a cost effective manner. Researchers do not need to be geometry experts in order to accomplish their mission. GEOLAB has become an interface between Langley Research Center organizations and their customers to allow rapid interchange and use of geometry data. GEOLAB is developing new technology for surface modeling and grid generation, and in addition to using the technology in-house, is transferring it to other NASA, government, university, and industrial organizations.

# References


[1] Vedat Akdag and Armin Wulf. Integrated geometry and grid generation system for complex configurations. *Proceedings of the Software Systems for Surface Modeling and Grid Generation*, pages 161–172, April-30,1992. NASA Conference Publication 3143.

[2] R. T. Biedron and J. L. Thomas. A generalized patched-grid algorithm with application to the F-18 forebody with actuated control strake. *Computing Systems in Engineering*, 1(2-4):563–576, 1990.

[3] F. Ghaffari, J. M. Luckring, J. L. Thomas, B. L. Bates, and R. T. Biedron. Multiblock Navier-Stokes solutions about the F/A-18 Wing-LEX-Fuselage configuration. *Journal of Aircraft*, 30(3):293–303, 1993.

[4] W. T. Jones and Jamshid Samareh-Abolhassani. A grid generation system for multi-disciplinary design optimization. *Proceedings of the 12th AIAA Computational Fluid Dynamics Conference*, June 1995. San Diego, California. AIAA Paper 95-1689.

[5] Paresh Parikh, Shahyar Pirzadeh, and Rainald Lohner. A package for 3-D unstructured grid generation, finite-element flow solution and flow field visualization. NASA Contractor Report 182090, September 1990.

[6] S. Pirzadeh. Recent progress in unstructured grid generation. January 1992. AIAA Paper 92-0445.

[7] M.-A. K. Posenau. Approaches to high aspect ratio triangulations. Technical report, NASA Langley Research Center, August 1993. NASA Technical Memorandum 107684.

[8] M.-A. K. Posenau and D. M. Mount. Delaunay triangulation and computational fluid dynamics meshes. Technical report, NASA Langley Research Center, August 1992. NASA Technical Memorandum 107663.

[9] Jamshid Samareh-Abolhassani. GridTool: Surface modeling and grid generation tool. *Workshop for Surface Modeling, Grid Generation, and Related CFD Issues*, May 9-11, 1995. Held at NASA Lewis Research Center.

[10] Jamshid Samerah-Abolhassani. Unstructured grids on NURBS surfaces. *Proceedings 11th Applied Aerodynamics Conference*, August 9-11, 1993. Monterey, California. AIAA Paper 93-3454.

[11] R. Smith, M. Bloor, M. Wilson, and A. Thomas. Rapid airplane parametric input design (RAPID). *Proceedings of the 12th AIAA Computational Fluid Dynamics Conference*, June 1995. San Diego, California. AIAA Paper 95-1687.

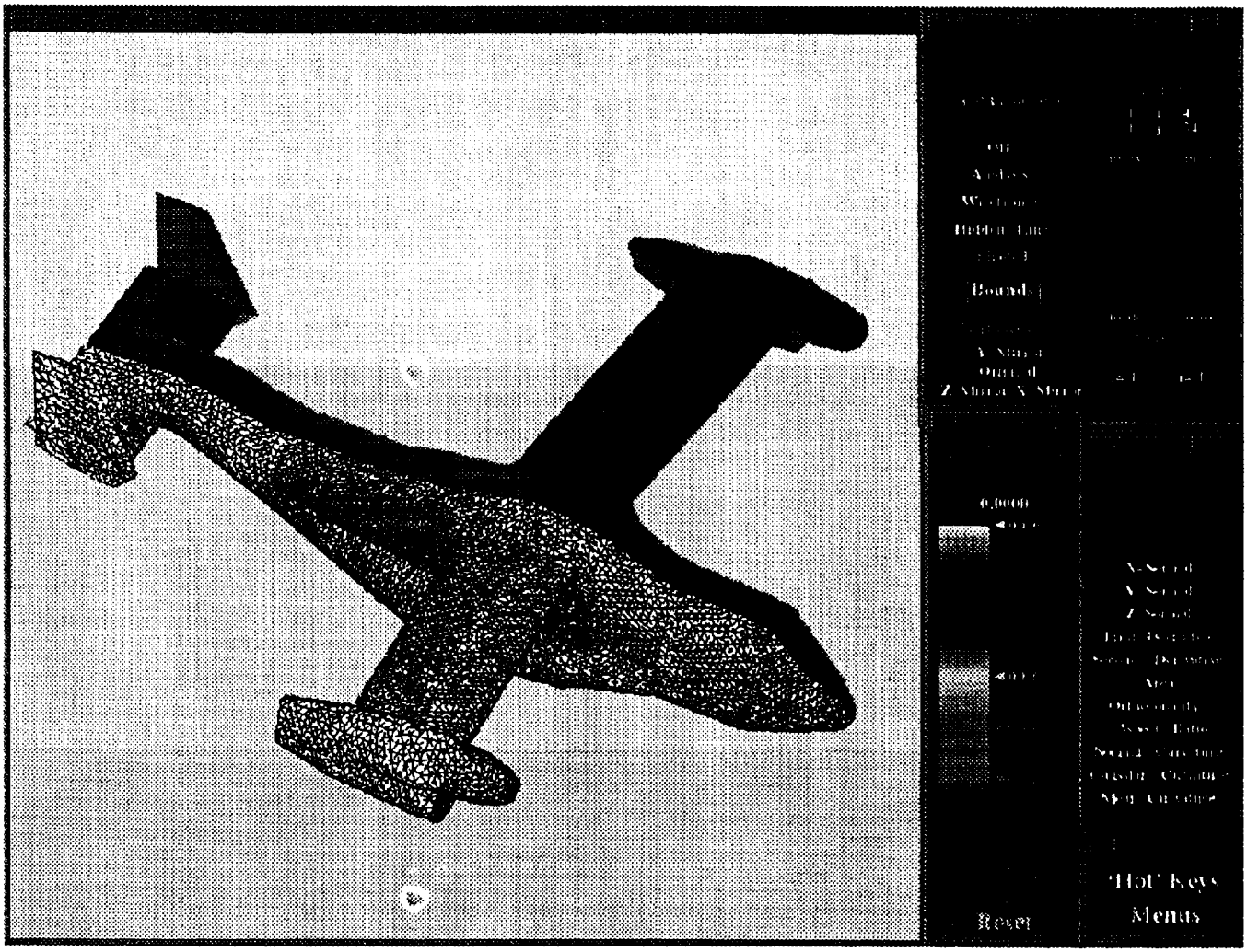
[12] John P. Steinbrenner and John R. Chawner. The GRIDGEN version 8 multiple block grid generation software. Technical report, MDA Engineering Inc., Arlington, Texas, December 1992. MDA Engineering Report 92-01.

[13] John P. Steinbrenner, John R. Chawner, and Chris L. Fouts. The GRIDGEN 3D multiple block grid generation system volume 1: Final report. Technical report, Wright Research & Development Center, Wright Patterson AFB, July 1990. WRDC-TR-90-3022.

[14] John E. Stewart and Jamshid Abolhassani. A graphically interactive approach to structured and unstructured surface grid quality analysis. *Proceedings AIAA/AHS/ASEE Aerospace Design Conference*, February 15-18, 1993. Anaheim, California. AIAA Paper 93-3351-CP.

[15] J. L. Thomas, S. L. Krist, and W. K. Anderson. Navier-Stokes computations of vortical flows over low-aspect-ratio wings. *AIAA Journal*, 28(2):205–2123, 1990.

Figure 1: An unstructured surface grid for the V22 Osprey is shown in SurfACe along with a shaded representation of the surface geometry.




ORIGINAL PAGE IS
OF POOR QUALITY

Figure 2: The surface geometry for the Apache helicopter is shown in GridTool with several patch boundaries and point distributions defined prior to projection to the surfaces.

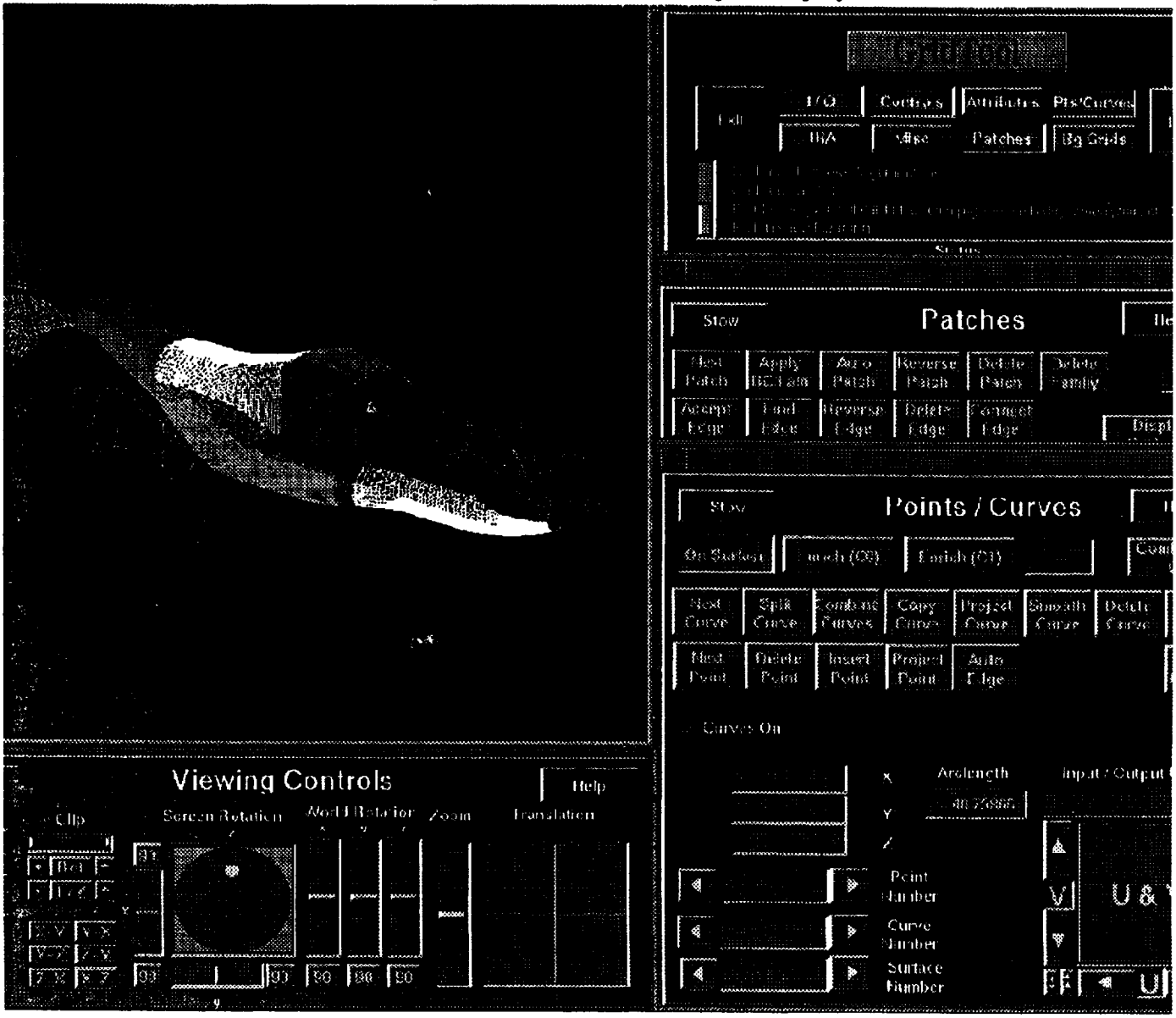
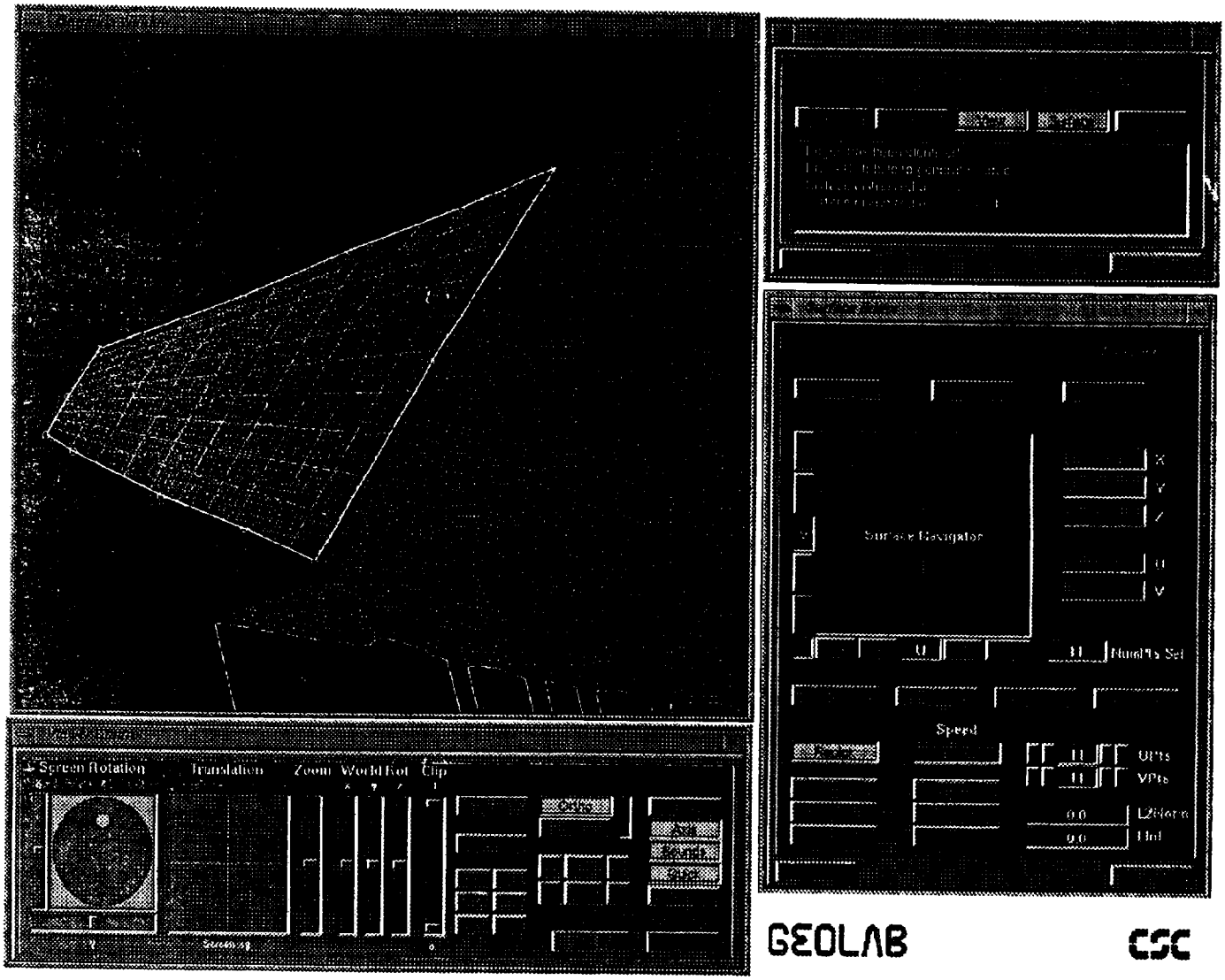
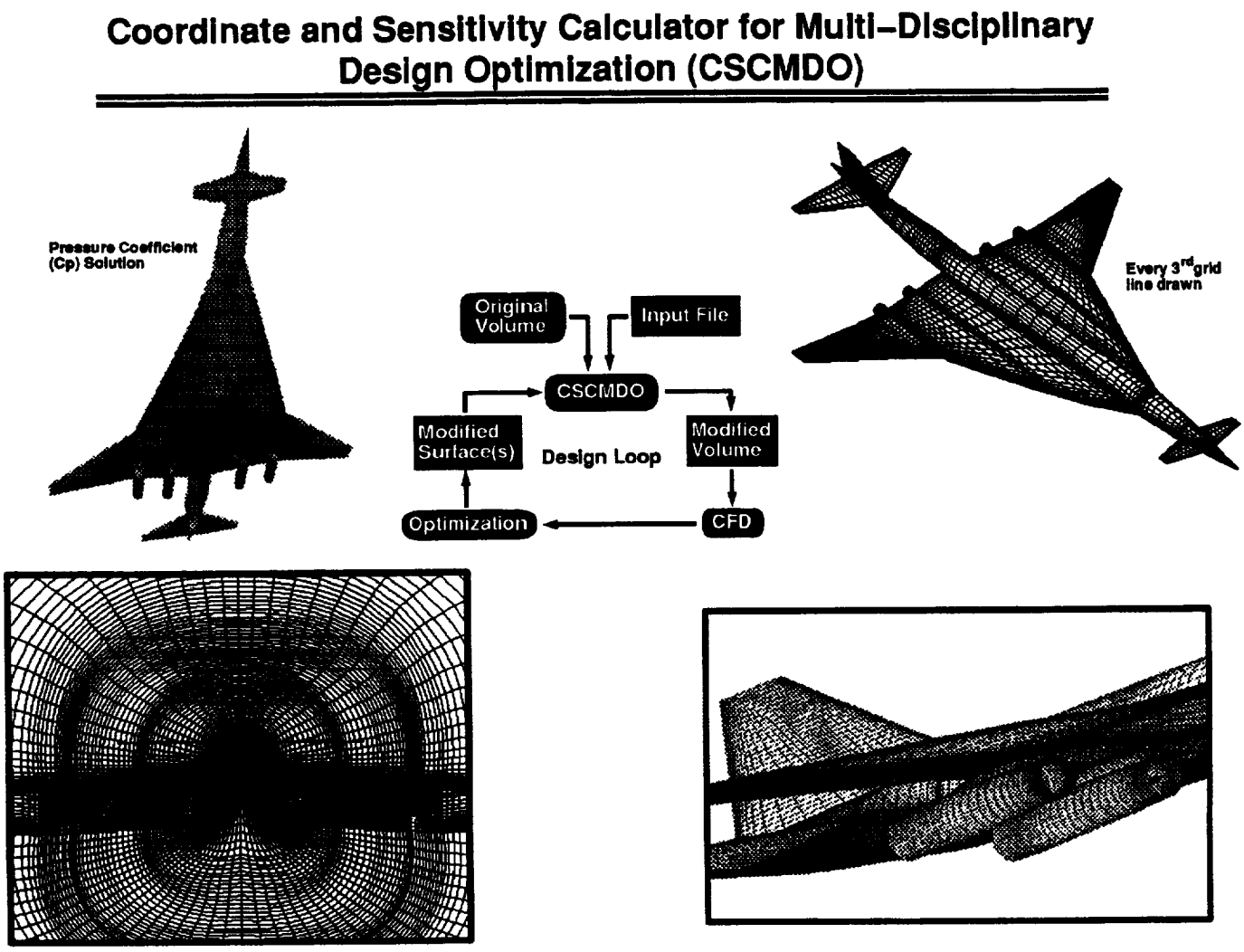


Figure 3: The measured points from the wing of an F-22 model is shown along with the newly fitted mathematical surface in the SCAFFOLD program.




ORIGINAL PAGE IS
OF POOR QUALITY

Figure 4: The integration of the CSCMDO program into the design loop is shown



# A Software Tool Summary

GEOLAB uses a variety of software for surface modeling, grid generation, and flow field validation of geometry. The purpose of the Appendix is to give the reader an overview of the software. The Commercial packages are:

- GRIDGEN
- ICEMCFD
- AZ2000/AZ3000
- PATRAN

Tools developed in GEOLAB available to the U.S. Government, Universities, and Industry are:

- CONVERT
- CSCMDO
- GridTool
- SCAFFOLD
- SurfACe
- VOLUME

Tools developed at Langley or other NASA facilities and available to the U.S. Government, Universities, and Industry are:

- VGRID/USM3D
- TLNS3DMB
- CFL3D

## A.1 GRIDGEN

GRIDGEN version 8 is a system of four codes for the generation of 3D, multiple block, structured grids: GRIDGEN version 8 was developed by MDA Corporation and sponsored by the Langley Research Center. The four programs are:

GRIDBLOCK is an interactive code for decomposition of the domain of interest into blocks, distribution of points along edges, and algebraic surface grid generation.

GRIDGEN2D is an interactive code for elliptic PDE refinement of 3D surface grids on the six faces of each block in the system.

GRIDGEN3D is a batch code for the generation of 3D volume grids on the interior of each block in the system.

GRIDVUE3D: Interactive code for inspecting volume grids.

The three interactive codes have been written using the Silicon Graphics IrisGL graphics library for both the user interface (e.g., buttons, text input) and the grid rendering. As such, these codes will only run on hardware that supports IrisGL. Version 8.3 of Gridblock, Version 8.4 of Gridgen2d, Version 8.8 of Gridgen3d, and Version 8.6 of Gridvue3d are currently available.

GRIDGEN version 9 integrates the functions of GRIDBLOCK and GRIDGEN2D into a single interactive program. GRIDGEN version 9 was developed by MDA corporation and sponsored by the AMES Research Center. GRIDGEN version 9 is available from COSMIC.

## A.2 ICEMCFD

ICEMCFD is a grid generation system built on top of a full CAD system. Geometry data can either be created within the system or read as either point format or IGES format. The grid is created independent of the geometry and at the end of the process projected directly onto the CAD surface geometry. This patch-independent approach can overlook small gaps and overlaps of the surfaces in the geometry.

ICEMCFD can be used to produce multi-block structured grids, unstructured tetrahedral grids, and body-fitted Cartesian grids. The resulting grids, topology and boundary conditions can be output in a number of formats to match different flow solvers that may be used.

## A.3 AZ2000/AZ3000

AZ2000/AZ3000 are software packages developed by Program Development Corporation for generating and displaying two and three dimensional multiblock structured grids. The packages automatically determine the blocking structure around complex geometries and accommodate nested grids. One license for the 3D 500 block, 1 million grid point code has been purchased, for assessment of code capabilities.

## A.4 PATRAN

P3/PATRAN is a general purpose three-dimensional Mechanical Computer-Aided Engineering(MCAE) software developed by PDA Engineering. P3/PATRAN provides a graphical environment where geometry can be modeled and pre-processed for analysis; the analysis results can be post-processed within the P3/PATRAN environment. It allows direct access to geometry from CAD systems such as Unigraphics, Pro/ENGINEER, and CATIA or as Initial Graphics Exchange Specification(IGES) entities. It also provides analysis preferences for code specific data input for analysis codes such as ABAQUS, ANSYS, MARC, and MSC/NASTRAN.

## A.5 CONVERT

CONVERT is a batch program that allows one to convert grids to/from various formats such as binary, formatted, unformatted, single precision, double precision, PLOT3D, GRIDGEN, LaWGS, Tecplot, etc.

## A.6 CSCMDO

CSCMDO is a general multi-block three dimensional volume generator suitable for multidisciplinary design optimization. The code is highly automated, robust, and efficient. Algebraic techniques are used to generate and/or modify block face and volume grids to reflect geometric changes resulting from design optimization. Volume grids are generated/modified in a batch environment and controlled via an ASCII user input "deck". This allows the code to be used directly in the design loop. The code has been written in ANSI "C" to be platform independent. CSCMDO has been tested extensively on aerospace configurations. The test cases range from simple wing/body configurations to full HSCT geometry with tail surfaces, engine nacelles, and canards. The code is also used outside the design loop in the GEOLAB for rapid modification and quality checks of existing CFD volume grids.

## A.7 GridTool

GridTool is an interactive program for IRIS workstations. This program has been developed for unstructured and structured grids. In unstructured areas, the code is capable of generating an input file for VGRID systems. Surfaces can be read either in point form such as GRIDGEN, PLOT3D, LaWGS, etc., or NURBS form such as IGES-128. Then, the surfaces are represented internally as NURBS surfaces. Also, the code can be used to project either unstructured or structured surface grids onto NURBS surfaces. There is a batch version available for projecting unstructured and structured surface grids.

## A.8 SCAFFOLD

SCAFFOLD is an interactive program that allows surfaces to be constructed from a group of points, such as surface measurements from a laser digitizer or cordax machine. The program accepts x, y, and z point values in the following formats: ECHO (laser digitizer), LaWGS, PLOT3D, and GRIDGEN. A new surface is created first by defining the points along the surface edge by selecting existing points. From the edge points, four edges are specified by selecting four corner points. An m x n mesh is defined by bilinearly interpolating points along the edges and in the interior of the new surface. The mesh is then projected onto the point sample using a normal projection algorithm to obtain the curvature of the original sample. The program also includes other tools to aid in surface development such as surface shading, displaying surface normals, and interactive means to rotate and translate surfaces.

SCAFFOLD is coded in the C programming language and runs on Silicon Graphics Iris workstations. The code is still under development with most of the effort concentrated on automating the process described above. Currently, work is being directed toward developing methods to extract geometry and topology from a group of points to form mathematically represented surfaces. SCAFFOLD is a simple solution to this problem since it provides the tools necessary to develop surfaces that can be manipulated by existing CAD/CAM packages, but it requires heavy user interaction and user knowledge of the topology represented by the sample points. More sophisticated algorithms can be easily incorporated into SCAFFOLD as they become available.

## A.9 SurfACe

A Surface Analysis Code (SurfACe) has been developed to help researchers assess surface grid quality of computational grids used in CFD analyses. Anomalies in grids used in these analyses can result in flow solutions that are not consistent with the true flow field characteristics of the vehicle. SurfACe can be used to highlight grid generation errors that are not easily detected in wireframe or shaded representations of a grid and thereby can increase the cost effectiveness of CFD as a design tool.

SurfACe can be used to evaluate both structured and unstructured surface grids on a number of grid quality parameters that indicate changes in surface curvature and changes in surface grid quality. Surface curvature parameters included related to grid smoothness are: the magnitudes of the x-, y-, and z-components of the surface normal vectors, first and second derivatives of these vectors, and the normal, Gaussian, and mean curvatures. Grid quality parameters included related to grid resolution are: surface grid cell area, orthogonality, and aspect ratio. Each parameter is displayed on the geometry using a variable color map. The displays can be viewed dynamically with rotation, translation, and scaling being controlled either by the keyboard or by the mouse. Wireframe, hidden line, and shaded views of the surface grid are also available.

## A.10 VOLUME

VOLUME is an interactive program written for IRIS workstations to generate multi-block structured volume grids. The code reads the surfaces of each block in either GRIDGEN or PLOT3D format. A transfinite method is used with the following blending functions: (1) Soni, (2) exponential, and (3) natural log. The unique feature of this program is the capability of not only specifying the boundary surfaces of each block but also the internal surfaces.

## A.11 VGRID/USM3D

The VGRID/USM3D aerodynamic analysis system is available for computing the flowfields around complex configurations. VGRID is a robust, user-oriented code for generating unstructured tetrahedral grids around very complex geometries by the Advancing Front Method. USM3D is an upwind flow code for solving the Euler equations on tetrahedral grids. Input for the system is facilitated through the GridTool utility developed by CSC Corporation and available through GEOLAB. The system is widely used and is supported by the Transonic Aerodynamics Branch (Contact: Dr. Neal Frink/804-864-2864).

## A.12 TLNS3DMB and CFL3D

Two Reynolds-Averaged Navier-Stokes solvers developed in the Computational Fluid Dynamics Laboratory (B1192) are available for computations on block-structured grids. The two codes, TLNS3DMB and CFL3D, can and have been used extensively for a variety of applications across the Mach number range. The features of the two codes are:

Steady and unsteady strong conservation law forms of compressible flows

Finite-volume discretizations

Euler and Navier-Stokes (laminar and Reynolds-averaged) solvers

Second-order spatial accuracy

Range of turbulence models from algebraic to two-equation models

Full MultiGrid (FMG) acceleration, including grid sequencing, to steady state

Perfect gas equation of state

The TLNS3DMB code has evolved from central-differencing concepts for the convective and pressure terms while the CFL3D code has evolved from upwind-differencing concepts. Both codes treat the viscous terms with central differencing. Either code allows an arbitrary

number of generalized coordinate blocks. The CFL3D code has generalized patched and overset grid capabilities while TLNS3DMB requires a one-to-one connection between the grid points of the blocks. A unified input and output format is currently being developed and tested for both codes. Contact: Dr. Chris Rumsey (804-864-2165) for CFL3D and Dr. Veer Vatsa (804-864-2236) for TLNS3DMB.

N95-28730

19951223 09

# B In-House Projects

## B.1 Rapid Airplane Parametric Input Design (RAPID)

Robert E. Smith
NASA Langley Research Center
Hampton, Virginia

### Abstract

RAPID is a methodology and software system to define a class of airplane configurations and directly evaluate surface grids, volume grids, and grid sensitivity on and about the configurations. A distinguishing characteristic which separates RAPID from other airplane surface modelers is that the output grids and grid sensitivity are directly applicable in CFD analysis. A small set of design parameters and grid control parameters govern the process which is incorporated into interactive software for "real time" visual analysis and into batch software for the application of optimization technology. The computed surface grids and volume grids are suitable for a wide range of Computational Fluid Dynamics (CFD) simulation. The general airplane configuration has wing, fuselage, horizontal tail, and vertical tail components. The double-delta wing and tail components are manifested by solving a fourth order partial differential equation (PDE) subject to Dirichlet and Neumann boundary conditions. The design parameters are incorporated into the boundary conditions and therefore govern the shapes of the surfaces. The PDE solution yields a smooth transition between boundaries. Surface grids suitable for CFD calculation are created by establishing an H-type topology about the configuration and incorporating grid spacing functions in the PDE equation for the lifting components and the fuselage definition equations. User specified grid parameters govern the location and degree of grid concentration. A two-block volume grid about a configuration is calculated using the Control Point Form (CPF) technique. The interactive software, which runs on Silicon Graphics IRIS workstations, allows design parameters to be continuously varied and the resulting surface grid to be observed in real time. The batch software computes both the surface and volume grids and also computes the sensitivity of the output grid with respect to the input design parameters by applying the precompiler tool ADIFOR to the grid generation program. The output of ADIFOR is a new source code containing the old code plus expressions for derivatives of specified dependent variables (grid coordinates) with respect to specified independent variables (design parameters). The RAPID methodology and software provide a means of rapidly defining numerical prototypes, grids, and grid sensitivity of a class of airplane configurations. This technology and software is highly useful for CFD research for preliminary design and optimization processes.



C-2

N95-28731

19951223/0

# B.2 Algorithms for High Aspect Ratio Oriented Triangulations

Mary-Anne K. Posenau
NASA Langley Research Center
Hampton, Virginia

## Abstract

Grid generation plays an integral part in the solution of computational fluid dynamics problems for aerodynamics applications. A major difficulty with standard structured grid generation, which produces quadrilateral (or hexahedral) elements with implicit connectivity, has been the requirement for a great deal of human intervention in developing grids around complex configurations. This has led to investigations into unstructured grids with explicit connectivities, which are primarily composed of triangular (or tetrahedral) elements, although other subdivisions of convex cells may be used. The existence of large gradients in the solution of aerodynamic problems may be exploited to reduce the computational effort by using high aspect ratio elements in high gradient regions. However, the heuristic approaches currently in use do not adequately address this need for high aspect ratio unstructured grids.

High aspect ratio triangulations very often produce the large angles that are to be avoided. Point generation techniques based on contour or front generation are judged to be the most promising in terms of being able to handle complicated multiple body objects, with this technique lending itself well to adaptivity. The eventual goal encompasses several phases: first, a partitioning phase, in which the Voronoi diagram of a set of points and line segments (the input set) will be generated to partition the input domain; second, a contour generation phase in which body-conforming contours are used to subdivide the partition further as well as introduce the foundation for aspect ratio control, and; third, a Steiner triangulation phase in which points are added to the partition to enable triangulation while controlling angle bounds and aspect ratio. This provides a combination of the advancing front/contour techniques and refinement. By using a front, aspect ratio can be better controlled. By using refinement, bounds on angles can be maintained, while attempting to minimize the number of Steiner points.

N95-28732

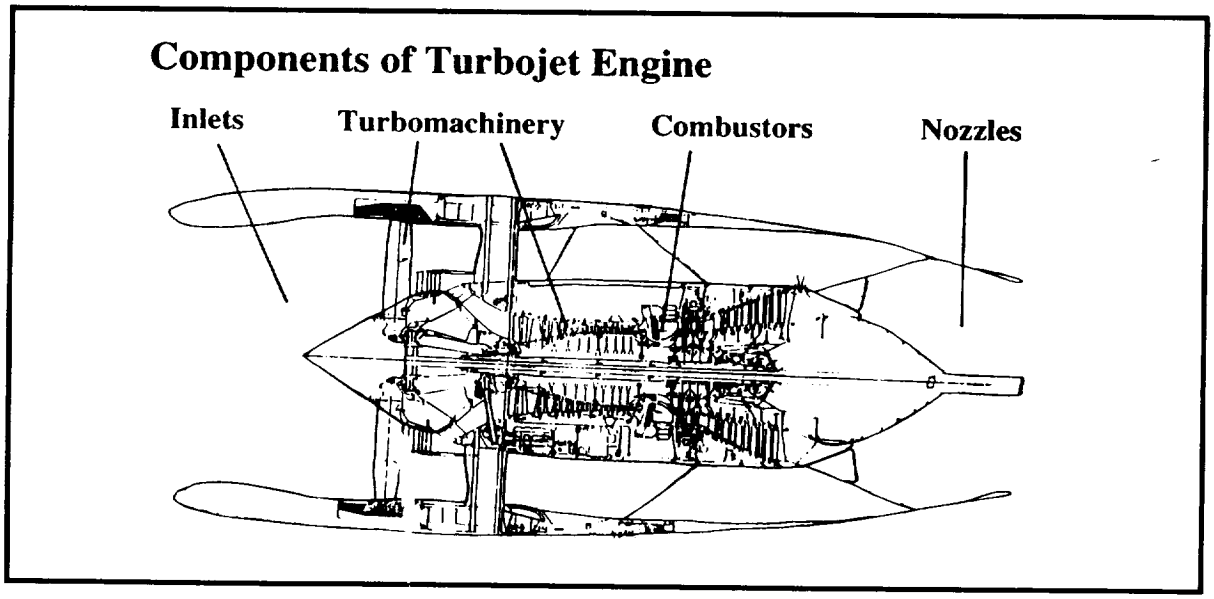
19951223 11

# SURFACE MODELING AND GRID GENERATION FOR AEROPROPULSION CFD

Yung K. Choo, [1] John W. Slater, [2] James Loellbach, [3] and Jinho Lee, [4]
NASA Lewis Research Center
Cleveland, Ohio 44135

## SUMMARY

The efforts in geometry modeling and grid generation at the NASA Lewis Research Center, as applied to the computational fluid dynamic (CFD) analysis of aeropropulsion systems, are presented. The efforts are mainly characterized by a focus on the analysis of components of an aeropropulsion system, which involve turbulent viscous flow with heat transfer and chemistry. Thus, this discussion will follow that characterization and will sequence through the components of typical propulsion systems consisting of inlets, compressors, combustors, turbines, and nozzles. For each component, some applications of CFD analysis will be presented to show how CFD is used to compute the desired performance information, how geometry modeling and grid generation are performed, and what issues have developed related to geometry modeling and grid generation. The discussion will illustrate the following needs related to geometry modeling and grid generation as observed in aeropropulsion analysis: (1) accurate and efficient resolution of turbulent viscous and chemically-reacting flowfields, (2) easy-to-use interfaces with CAD data for automated grid generation about complex geometries, and (3) automated batch grid generation software for use with design and optimization software.




[1]Senior Scientist, CFD Branch, Internal Fluid Mechanics Division
[2]National Research Council Associate, CFD Branch, Internal Fluid Mechanics Division
[3]Research Associate, Institute for Computational Mechanics in Propulsion (ICOMP)
[4]Senior Engineer, NYMA, Inc.

# Inlets

## NASA Lewis inlet analysis from subsonic to hypersonic flight

## Inlet flow characteristics to compute include:

**Pressure Recovery**
**Distortions at compressor face**
**Cowl drag**
**Flow quantity and engine matching**
**Flow stability**
**Flow angularity**

## Surface Modeling / Grid Generation / CFD:

**CAD software for surface modeling**
**NASA-IGES for data exchange**
**Interactive domain decomposition for multi-block grids**
**Space-marching, reduced-Navier-Stokes solvers on structured grids**
**Time-dependent, Navier-Stokes solvers on multi-block, structured grids**
**CFD codes are characterized by specialization to propulsion flowfields**

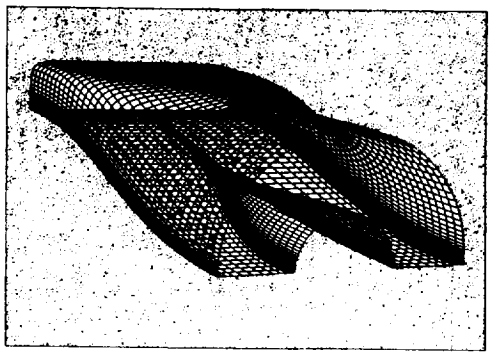
The purpose of the inlet is to provide the desired quantity and quality of airflow to the engine for stable production of thrust. The analysis of inlets at the NASA Lewis Research Center encompass flight speeds from subsonic to hypersonic.

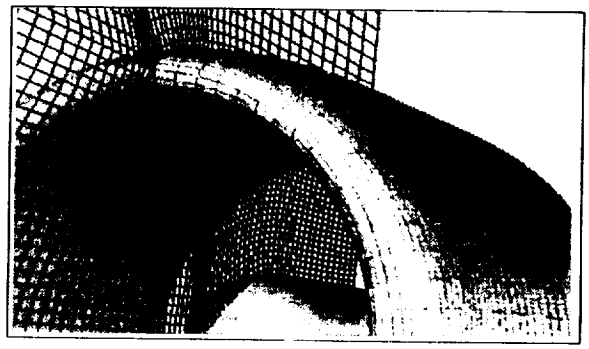
The CFD analyses are applied to determine such information as the mass flow rates, pressure recovery, distortions at the compressor face, inlet drag, flow angularity, and flow stability [1]. Most analyses involve steady-state flow; however, some unsteady flow analyses are conducted when examining inlet stability.

The primary CFD codes used at the NASA Lewis Research Center for inlet analysis include RNS3D [2] and NPARC [3]. The RNS3D code solves the reduced Navier-Stokes equations to efficiently model the flow using a space-marching method. The NPARC code solves the full, unsteady Navier-Stokes equations using time-marching methods. Both codes have undergone development to focus their abilities towards the analysis of propulsion flowfields, which are dominated by turbulent viscous effects. The codes include capabilities for bleed boundary conditions, turbulence models capable of predicting separation of flow from inlet walls, and compressor face boundary conditions. Both codes use structured grids primarily because of familiarity with the performance of the numerical methods and the desire to efficiently and accurately model turbulent viscous flows. The NPARC code uses multi-block, structured grids to model complex geometries. The blocks may overlap in a non-contiguous manner for difficult geometries.

## Inlet Shape Design and Optimization

**Bifurcated Transitioning S-Ducts** **Subsonic Transport Inlet Optimization**





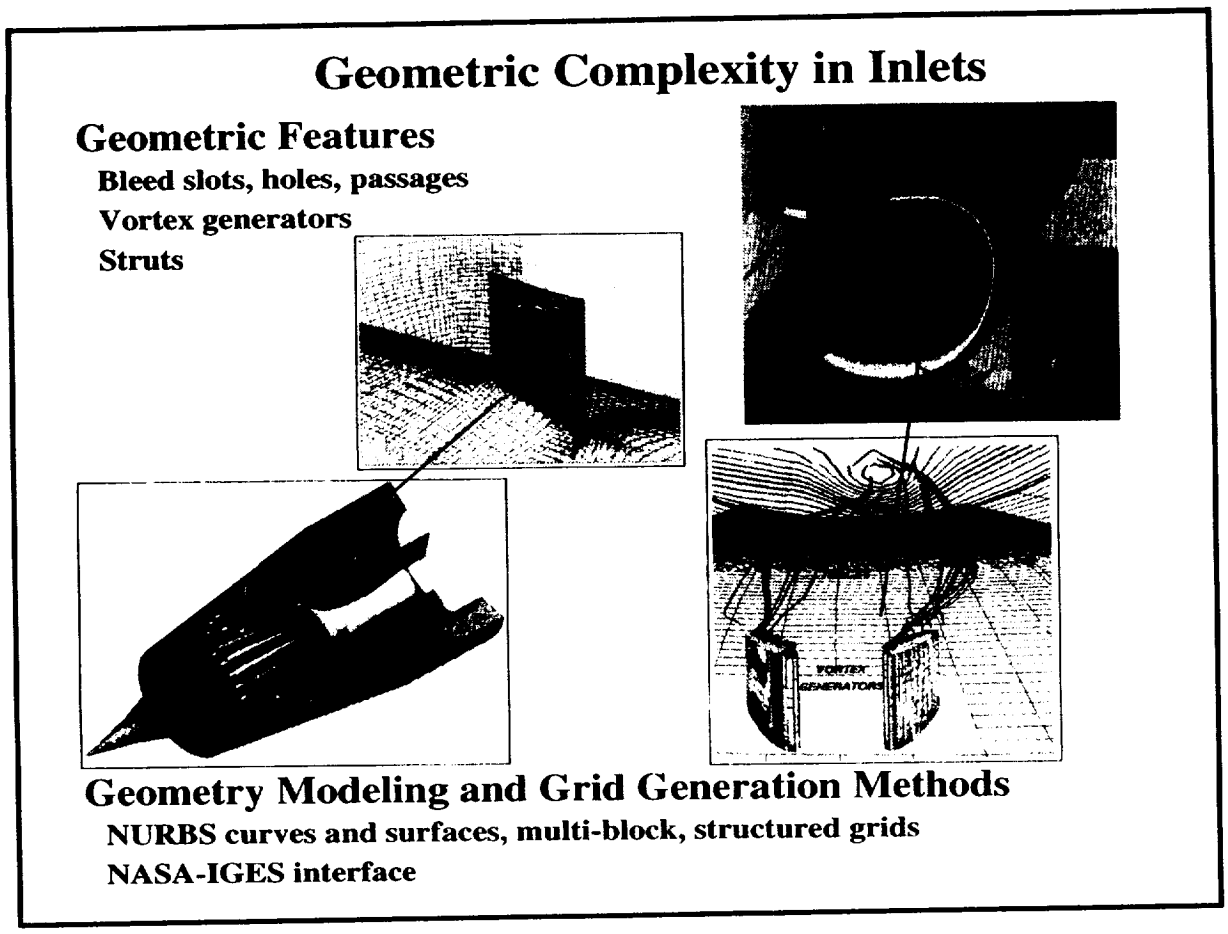
### Use of CFD in design environment

**Geometry defined by geometric parameters**
**Desire grid generation to be batch operation**
**Grid generation software as callable libraries**

Two applications of CFD methods to inlet shape design and optimization studies are presented to illustrate how geometry is modeled parametrically and grids are generated as batch processes. This requires grid generation software to be accessible in a format callable by the design software.

The first application is the analysis of bifurcated transitioning S-ducts for high-speed transport applications [4]. The analysis used the RNS3D and NPARC codes to compute pressure recovery and distortion at the compressor face. It is desirable to understand the relationship between the turbulent flow patterns and the shape of the cowl surface and the length of the inlet. The shape of the cowl was defined by parameters of an analytic equation. The structured grid was computed automatically with desired stretching for viscous resolution.

The second application is the shape optimization of a subsonic inlet [5]. The analysis modeled a 180 degree section of the inlet from the crown to the keel assuming symmetric geometry and flow conditions. At each section of the cowl, the profile was defined by a set of parameters of a superelliptic equation. The parameters varied as a function of the section angle. A three-dimensional grid was generated as a sequence of two-dimensional grids generated at each section angle of the inlet using an elliptic grid generation method from GRAPE2D. The NPARC code was used to compute the Mach number distributions on the surfaces of the inlet. The objective was to optimize the shape of the interior of the cowl such as to minimize the peak Mach number within the inlet. The optimization loop required the grid generation to be a batch process to allow regeneration of the grid at each optimization iteration.




An inlet may consist of several complex geometric features, including bleed slots and holes, bleed passages, vortex generators, and struts. The struts are present due to structural and ducting requirements. The scale of the bleed slots and holes and the vortex generators are often small compared to the size of the duct, but their influence on the inlet flowfield may be quite significant. They are usually introduced to stabilize and to improve the flow. Because of their significance to the flow, an analysis usually includes their pressence either numerically or physically. RNS3D includes a numeric representation of a set of vortex generators as vortex sources in the flow. NPARC includes flow boundary conditions to simulate bleed surfaces.

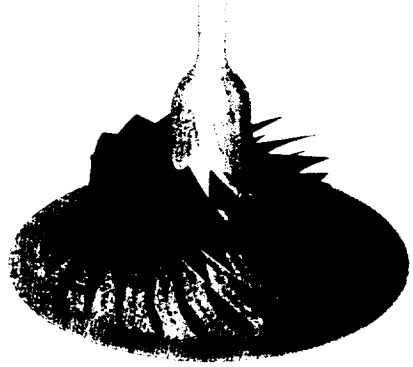
Some studies have begun to physically model bleed features and vortex generators as a way to bypass numerical modeling of these features. The figures above show some examples of grids generated about inlet geometries that include vortex generators. The high-speed inlet shown above made use of the capabilities of the ICEM DDN software to create a geometry model of the inlet from blueprint drawings [6]. Such a geometry model may come from a CAD effort. The ICEM CFD software was then used to create a multi-block, structured grid about the vortex generator. Some compromise of the vortex generator tip geometry was needed to simplify the multi-block format.

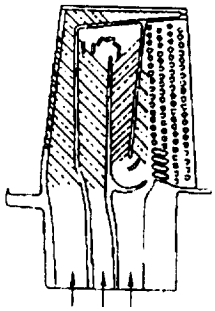
The F/A-18 inlet analysis shown above used the I3G/VIRGO and GRIDGEN software to create the geometry and multi-block, structured grid from axial station coordinates [7]. The turbulent viscous flow analysis examined the flow in the inlet for high angle-of-attack flow conditions. The analysis includes the geometric modeling of a pair of vortex generator within the duct.

## Turbomachinery Geometry Issues

**Primary geometric features, such as endwall and blade shapes, are accurately represented.**

**Secondary features, such as cooling holes and seals, are often ignored to reduce cost.**



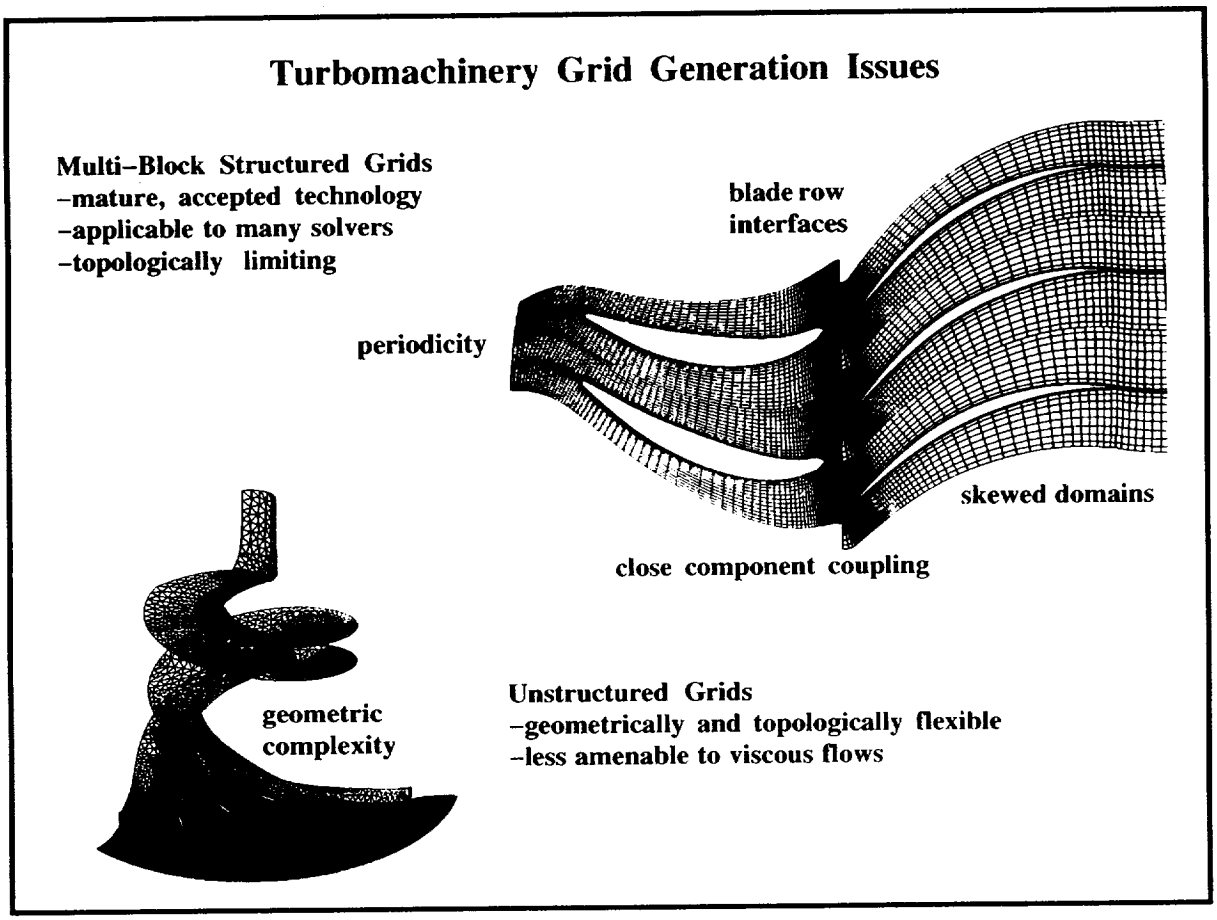


**The cost of accurately representing small geometric details is very high, from both grid generation and flow solution standpoints.**

Turbomachinery refers to a broad class of devices which transfer energy to or from a working fluid by the use of rotating components. Examples include nonreciprocating pumps and compressors, and turbines. Turbomachinery research at NASA Lewis is focused on applications to aircraft propulsion (radial and axial compressors and axial turbines in turbojet engines) and rocket propulsion (radial pumps and axial turbines in liquid fuel turbopumps). The objective of any turbomachinery component is to accomplish the desired transfer of energy to or from the working fluid with as little loss as possible in order to maximize the overall efficiency.

The typical level of geometric modeling used in numerical analyses of turbomachinery components consists of accurate representations of the channel boundaries (hub and tip endwalls) and the blade shapes. Small geometric features such as coolant holes and passages, tip clearance gaps, and seals between rotating and nonrotating components are often ignored in the interest of reducing the solution cost. The fluid-dynamic effects of certain small geometric details can sometimes be numerically modeled in the flow solver without accurate geometric representation, but usually to only a low order of accuracy. When the fluid dynamic effects of the small geometric details are of primary importance, they must be included in the geometric model. This is very costly, from both the grid generation and flow solution standpoints. Automated methods for topology determination and grid generation could be of use for such geometric complexity.

Geometry modeling techniques commonly used for turbomachinery configurations include CAD-based methods, NURBS representations, or simple discrete point representations of surfaces used in conjunction with different surface fitting and interpolation schemes. There currently is no real standard for geometry modeling of turbomachinery components.




A variety of grid types have been used in turbomachinery component analysis, ranging from single-block and multi-block structured grids to fully unstructured grids. Common structured grid topologies for turbine and compressor blades include C-, H-, and O-meshes. Multiple blocks are required for all but the simplest of configurations. Although the use of unstructured grids for turbomachinery analysis is increasing, the use of structured grids remains somewhat more common. This is due to the generally lower computational resource requirements of structured flow solvers and the availability of techniques for generating highly-stretched structured grids for viscous flows.

It is often only necessary to generate a grid for a single blade in a given row; the grids for the other blades in the row are copied and rotated from the original grid. Smooth transmission of flow properties across the boundaries of adjacent grids requires either that the grids be periodic or that the flow solver account for nonperiodic grids. Blade rows which rotate relative to each other are usually separated by a sliding grid interface, which therefore must be axially symmetric. These constraints, when applied to closely coupled, high-solidity blade rows, often result in computational domains with high inherent skewness and make the generation of nearly-orthogonal structured grids difficult. For viscous calculations, grid points should be clustered normal to solid walls and in the blade wake regions. This is difficult to achieve in unsteady, multiple blade row calculations in which the wakes of rotating components move with time. The use of unstructured grids can alleviate some of the problems mentioned above due to their greater geometric flexibility. Hybrid techniques combine both structured and unstructured grids in a single calculation in an attempt to exploit the advantages of each approach.

## Turbomachinery Flow Solution Issues

**Actual flows are unsteady and strongly affected by turbulence and viscosity.**

**Steady–state analyses of isolated components can usually be performed in body–fixed, rotating coordinate systems.**

**Mixing–plane interfaces can be used to couple components when unsteady interactions are not required.**

**Otherwise, time–accurate, unsteady, simultaneous solution techniques must be used for multiple components.**

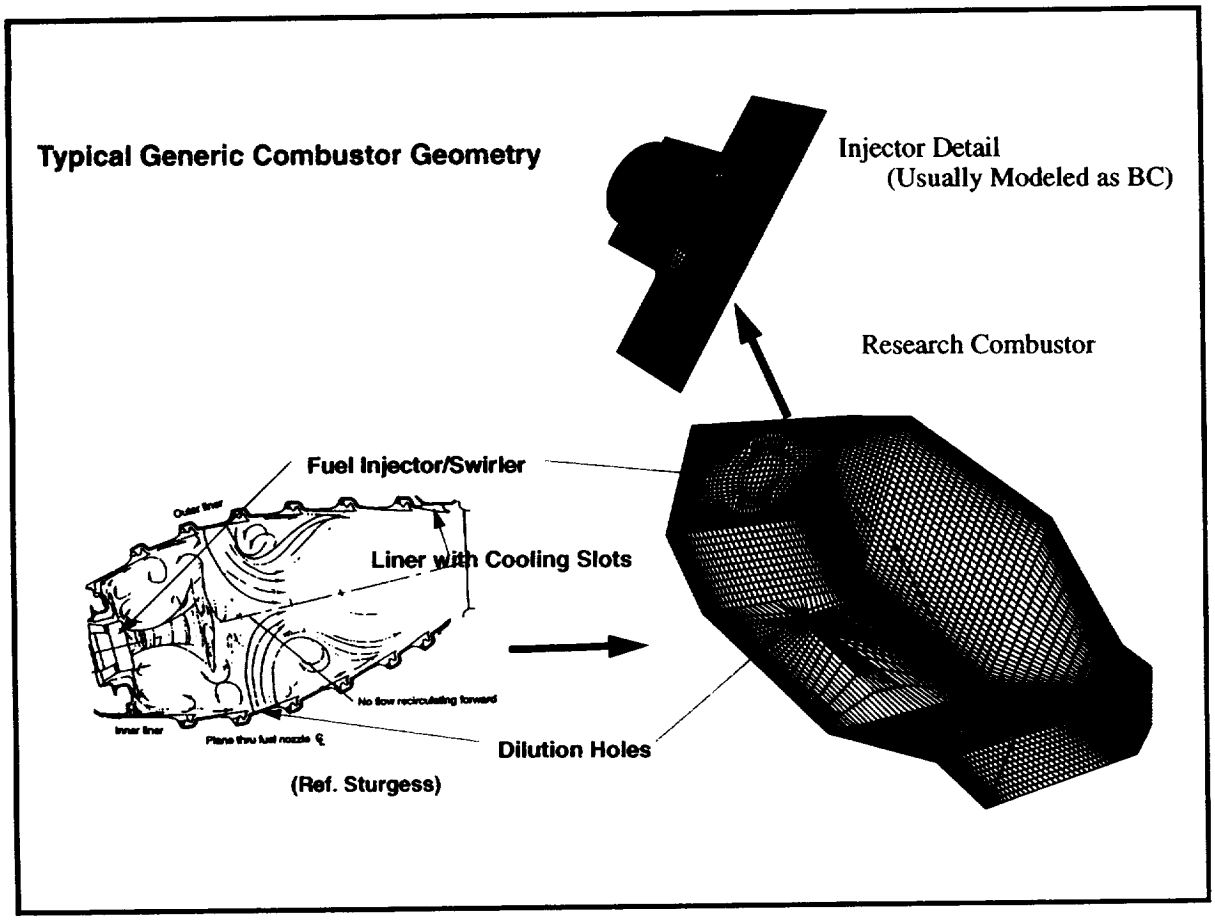
**Several 3D Navier–Stokes solvers have been developed specifically for turbomachinery problems.**

**The choice of solver depends on what problem is being solved, and what information is desired from the solution.**

Actual turbomachinery flows are unsteady due to the relative rotation of components. They are also strongly affected by viscosity and turbulence because of the influence of upstream components and the internal nature of the flow. Typical engineering values of interest include blade loadings, loss distributions, and heat transfer rates. Time-averaged values are useful for overall performance evaluation while unsteady values are used to determine peak loads and temperatures. Blade loadings can sometimes be accurately predicted using Euler methods. Accurate loss and heat transfer predictions, however, require Navier-Stokes techniques free from excessive numerical diffusion, and grid systems appropriate to the flow problem and solver.

The analysis of isolated turbomachinery components can often be performed in a steady manner by using body-fixed, rotating coordinate systems. Mixing-plane interfaces can be used to transmit tangentially-averaged solution data between components when unsteady interactions are not significant. When such interactions are significant, time-accurate, unsteady, simultaneous solutions for the components are required.

Several three-dimensional, Navier-Stokes solvers are used at NASA Lewis for turbomachinery analysis. These include the structured solvers HAH3D [8], RVC3D [9], TRAF3D [10], MSTAG [11], and BTOB3D [12] and the unstructured solver USM3D [13], among others.

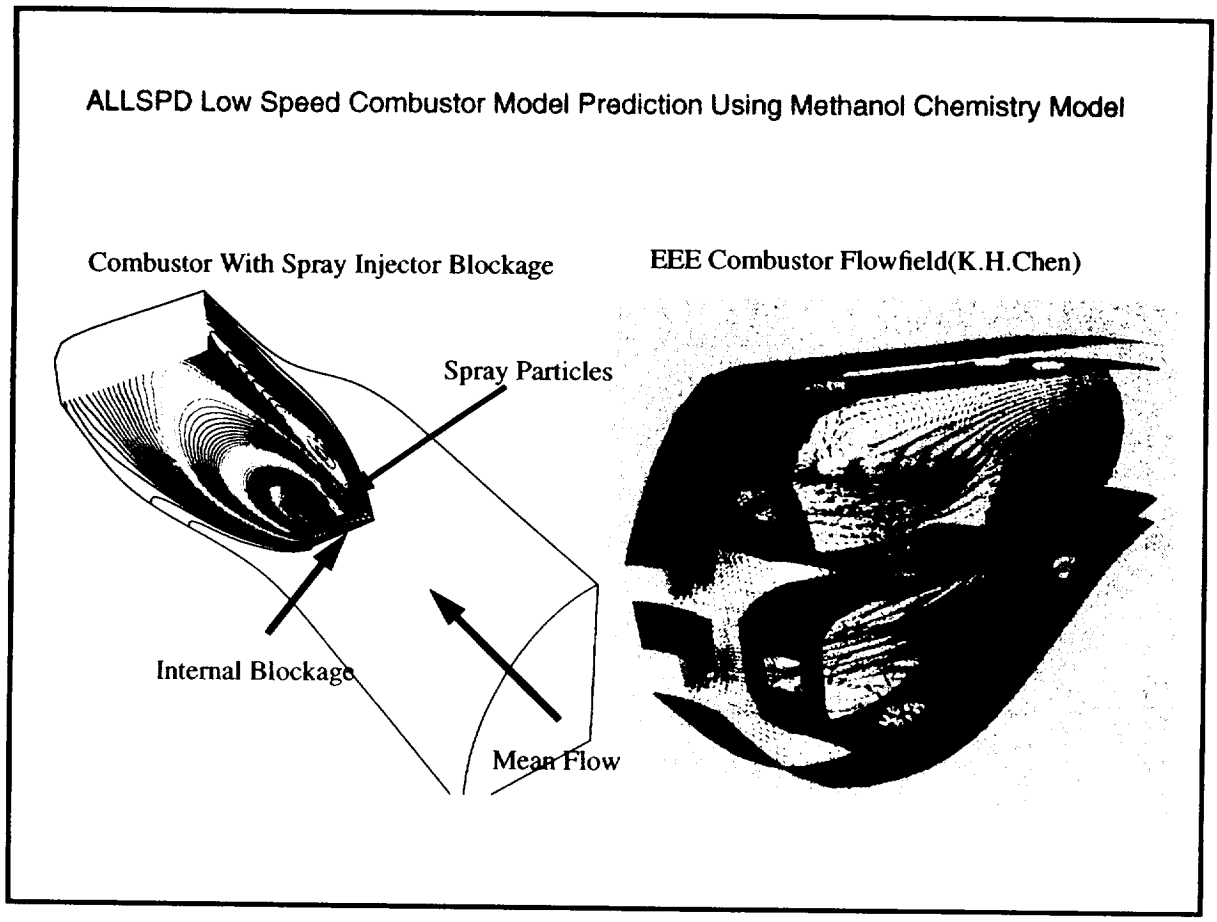



In any aeropropulsion system, the chemical potential of the fuel is converted into momentum gain in the combustor section. The next generation of commercial subsonic and supersonic aircraft require engines whose performance must be substantially better than current propulsion systems. This requires higher operating pressures and temperatures for the combustor. In addition, these engines must be capable of high performance with lower emissions levels to satisfy more stringent environmental regulations. Furthermore, these problems need to be resolved at a lower cost and in a shorter time frame.

Today's propulsion systems use very complex combustor and fuel injector geometries to obtain high combustion and mixing efficiencies for maximum gain of thrust from a given chemical potential. A typical gas turbine engine combustor geometry consists of a number of combustor "cans" arranged in annular form. The cannister illustrated in cross section above, contains a fuel injector with fuel atomizer, swirler, dilution holes, combustor liner with transpiration cooling holes, and slots with coolant injectors.

Combustor analysis requires the prediction of (HC, CO, NOx) emission characteristics, exit pattern factors, liner wall temperature levels and gradients, ignition, smoke and soot production, flame stability, flame blowout and relight.

The development of a new combustor design is often very difficult and costly. Present design analysis methods consist largely of one- or quasi-two- dimensional empirical and semi-empirical analyses, previous experience, and a large number of expensive component testing with a long development cycle time [14] [15].

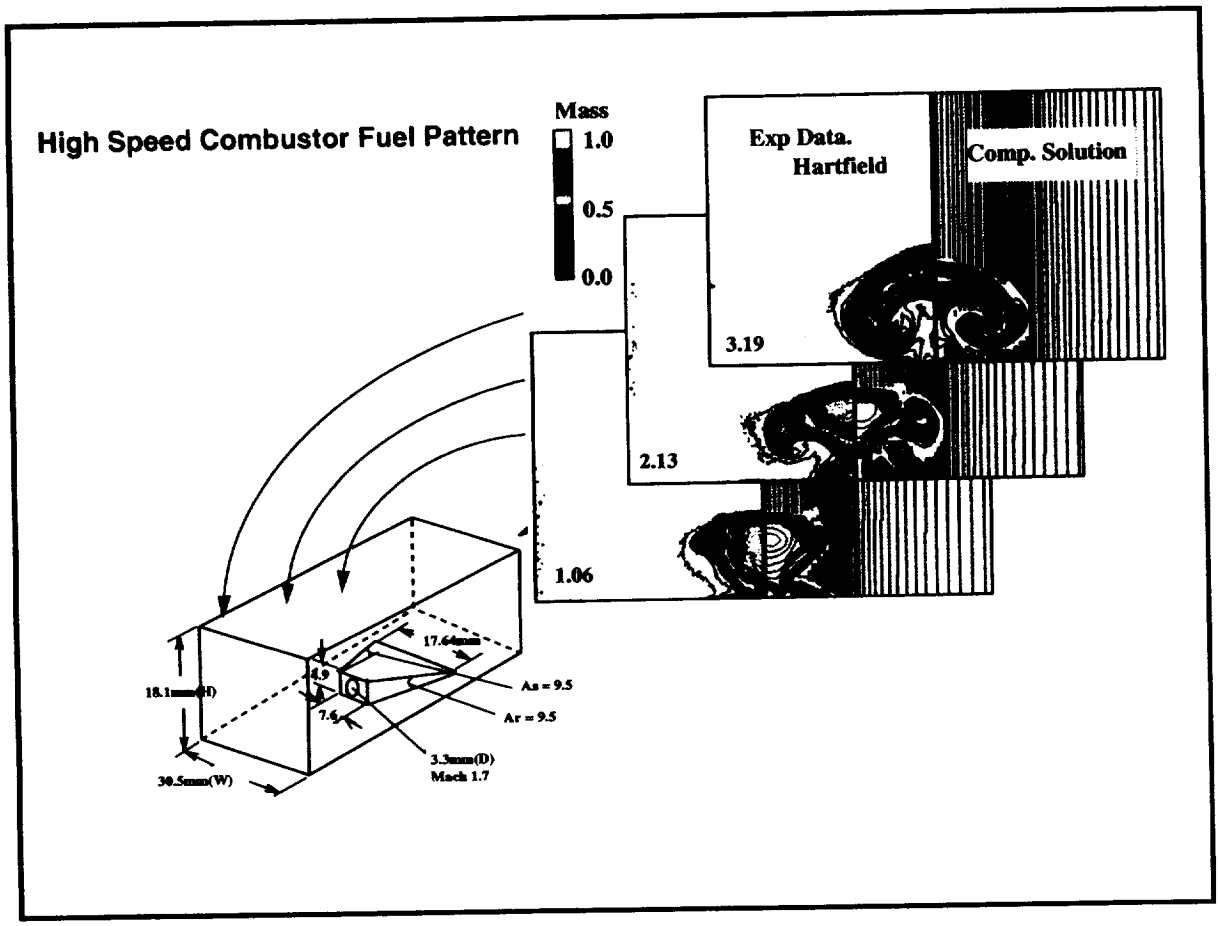



An improved analytical tools are needed to predict the performance of the combustor for complex combustor geometries. Design engineers would like to have analytical tools that contain the optimal balance of the best possible physics. These include accurate turbulence and chemistry models, fast and accurate numerical methods, the ability to handle complex geometries rapidly, and the capability to interface with other analytical tools.

In order to satisfy these challenges, fast and accurate analytical CFD tools are being developed at the NASA Lewis Research Center with refined physical models for combustor design. Two implicit LU solvers for reacting flows have been developed. The RPLUS code [16] is being developed for high-speed combustor flowfield analysis. The ALLSPD code [17] is being developed for low-speed combustor analysis. Both of these codes use the implicit treatment of the chemical source terms and have the capability to handle detailed finite-rate chemistry models, global chemistry models, Monte-Carlo PDF models [18], a low Reynolds number two-equation turbulence model, a spray chemistry model, and multiple-block structured grid systems.

The capability of the ALLSPD code is illustrated by the prediction of the temperature field inside of generic low-speed combustor shown above. A low-speed, low Reynolds number transition duct is used as a combustor, with combustion simulated using a methanol spray chemistry model. The figure also shows the flow pattern predicted by ALLSPD for the EEE combustor. Some of the more complex geometry, such as the combustor liner, is also modeled along with the spray chemistry.

An unstructured combustor solver with the majority of the physical model features contained in the ALLSPD code is also being developed.




The capability of the RPLUS code to predict the complex fuel injection patterns for a chemically reacting flowfield is shown above. The CFD prediction of the fuel penetration pattern generated by a swept ramp fuel injector is shown along with the experimental data. The fuel penetration and mixing is key to understanding and controlling combustion behavior.

For combustor analysis, additional chemical reaction / species transport equations are needed for the chemistry models, which can easily triple the number of equations which must be solved by the CFD codes. Three-dimensional combustor flowfield analysis with detailed chemistry models, until recently, has been limited to basic research efforts involving simplified geometries. The stiffness generated by the chemical time scales further restricts the convergence rate, which also restricts the size of the problem.

Research is being conducted to understand and to reduce the impact of uncertainties in the numerical modeling of turbulence, chemistry, and unresolved geometrical features. The combustor geometrical features such as fuel injectors and swirlers must either be numerically modeled as boundary conditions or resolved physically. Presently, the majority of these complexities are modeled as simple boundary conditions in order to reduce the computational cost.

The RPLUS and ALLSPD codes can be used with any structured grid generation software, such as CFD GEOM, GRIDGEN, NGP, GRIDPRO, and INGRID3D. The prevalent use of multi-block, structured grids has been due to the available wealth of mature numerical technology. As illustrated, the complex combustor geometry can be effectively treated using a structured multiple block grid topology with generalized block interface treatment.

Regardless of the topology, these codes will require: refined near-wall grid resolution with low cell aspect ratio, holes with various geometrical configurations, and smooth and nearly orthogonal grids. In addition, flexible internal grid structures for specification of blockages with irregular geometrical shapes fixed in space are also highly desirable.

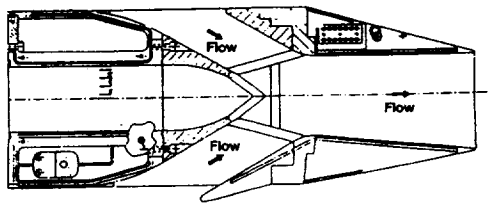
The analysis of next generation combustor designs requires that the CFD solvers have either the ability to accurately model or to numerically resolve the following key combustor design features: swirlers, fuel injectors, canister shapes with slots, dilution holes, cooling slots, and perforated walls.

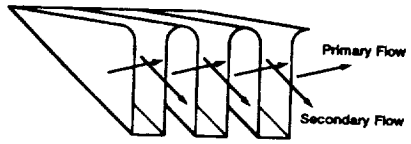
# Nozzles

## NASA Lewis nozzle analysis from subsonic to hypersonic flight

## Nozzle flow characteristics to compute include:

**Thrust**
**Flow mixing**
**Boattail drag**
**Noise**




## Mixer/Ejector Nozzle
## Surface Modeling / Grid Generation / CFD

**Time-dependent, Navier-Stokes solver on multi-block, structured grids**
**Interactive domain decomposition for multi-block, structured grids**

The purpose of the nozzle is to efficiently convert the thermal energy of the combustion products into the kinetic energy of the exhaust jet and to straighten the gas flow. The analysis of nozzles at the NASA Lewis Research Center encompass flight speeds from subsonic to hypersonic.

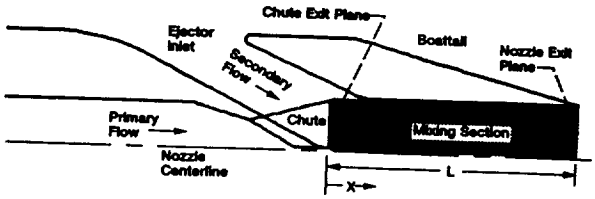
The CFD analyses are applied to determine such information as the thrust, boattail drag, and level of flow mixing of primary and secondary flows. The flows are dominantly turbulent viscous flowfields. The primary CFD code used for nozzle flow analysis is the NPARC code.

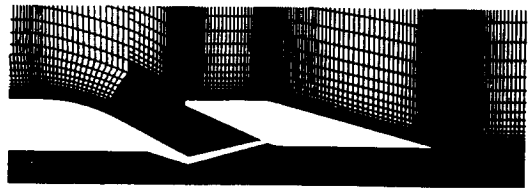
One example of the type of nozzle flow analysis being conducted at the NASA Lewis Research Center is the flow in a mixer/ejector nozzle during take-off and landing conditions [19]. Secondary flow is entrained into the primary jet flow through a set of lobed chutes introduced into the primary flow. The chute forms vortices which aid in the flow mixing. After take-off, the chutes retract back into the nozzle walls. The objective is to reduce the jet noise during take-off and landing. An application of such a nozzle would include high-speed civil transports.

Another significant problem under analysis is the computation of the boattail drag at transonic flow conditions. The objective is to generate a database of drag data for varying flow and geometric parameters.

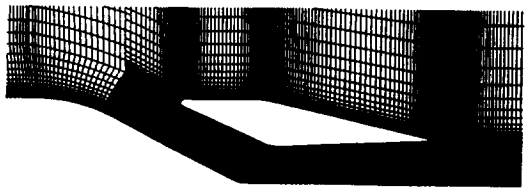
# Geometry and Grids for Mixer/Ejector Nozzle

## Mixer/Ejector Nozzle



(a) Primary centerline ($A_s/A_{mix} = 1.0$).



(b) Secondary centerline ($A_s/A_{mix} = 1.2$).

**Geometry read from detailed drawings**
**Geometry simplified to include only flow passages**
**Flow domain is 1/2 wavelength of chute lobes**
**8 structured grid blocks, clustered for turbulent flow computation**
**Interactive domain decomposition and grid generation**
**Grids include about 1 million grid points**

The mixer/ejector nozzle represents the typical geometric complexity of CFD analyses of nozzles. For the analysis discussed in reference [19], the geometric dimensions for the nozzle were read directly from detail drawings. Mechanical elements such as acuators, hinges and seals were not included in the geometry model. The flow domain consisted of 1/2 of the wavelength of a lobe of the chute with assumptions of flow symmetry. The I3G/VIRGO software package was used to define the surface geometry and grids. The INGRID3D software package was used to generate the volume grids. The grid contained 8 blocks with a total of about 1 million grid points. The grid used an H-grid topology. The grid was clustered near the walls for resolution of turbulent flow. One difficulty in generating the structured grid for the block containing the lobe of the chute was excessive grid skewing due to the S-shaped turn. This grid is typical of those used in the analysis of nozzle flows. The GRIDGEN software package is also used for grid generation.

## Concluding Remarks

Some concluding remarks that can be stated from this overview of the geometry modeling and grid generation from the perspective of the CFD analysis of propulsion components include the following:

- The geometry modeling and grid generation activities at NASA Lewis are oriented to the application to components of aeropropulsion systems.
- The aeropropulsion flowfields are dominantly turbulent viscous flowfields with heat transfer and chemical reactions.
- The complexity of the geometry and flows and limitations of computational resources require a compromise between numerical and physical modeling.
- The majority of the analyses are performed using structured grids because of the maturity of the CFD methods as applied to the computation of viscous flows with and without chemical reactions.
- As computational resources increase, the capability to include greater geometric complexity improves. This requires an easy-to-use interface with CAD representations and more automation in grid generation.
- Automated grid generation must still allow proper control over the grid distribution to properly resolve the flowfield.
- Design and optimization processes use CFD and grid generation in a batch mode. Interactive-based software, perhaps, can include an option for either the use of a graphical user interface or the use of callable libraries.

## REFERENCES


1. Anderson, B.H. and C.E. Towne, "Application of Computational Fluid Dynamics to Inlets," Practical Intake Aerodynamic Design, Ed. E.L. Goldsmith and J. Seddon, AIAA, Washington, D.C., 1993.
2. Towne, C.E., "RNS3D User's Guide: Version 0.8," January 1994.
3. NPARC Alliance Technical Team, "A User's Guide to NPARC: Version 2.0," November 1994.
4. Anderson, B.H. and K. Kapoor, "A Study on Bifurcated Transitioning S-Ducts for High Speed Inlet Application," AIAA-94-2812, June 1994.
5. Reddy, E.S. and D.R. Reddy, "Aerodynamic Shape Optimization of a Subsonic Inlet Using 3-D Euler Computation," Paper to be presented at the 31st AIAA/ASME/SAE/ASEE Joint Propulsion Conference, July 1995.

6. Slater, J.W., Joongkee Chung, Y.K. Choo, "Geometry Modeling, Grid Generation, and Flow Computations for Complex Geometries," AIAA-94-2717, June 1994.

7. Smith, C.F. and S.D. Podleski, "Installed F/A-18 Inlet Flow Calculations at 30 Degree Angle-of-Attack: A Comparative Study," AIAA-94-3213, June 1994.

8. Hah, C., "Calculations of Three-Dimensional Viscous Flows in Turbomachinery With a Low Reynolds Number Turbulence Model," AIAA Journal of Propulsion and Power, Vol. 3, No. 5, May 1987, pp. 415-422.

9. Chima, R.V, "Viscous Three-Dimensional Calculations of Transonic Fan Performance," NASA TM 103800, May 1991.

10. Arnone, A., Liou, M-S., and Povinelli, A., "Transonic Cascade Flow Calculations Using Non-Periodic C-type Grids," CFD Symposium on Aeropropulsion, NASA Conference Publication 3078, 1990.

11. Adamczyk, J.J., "Modeling Equation for Simulating Flows in Multistage Turbomachinery," ASME Paper 85-GT-220, 1985.

12. Dawes, W.N., "Development of a 3D Navier-Stokes Solver for Application to all Tyles of Turbomachinery," ASME Paper 88-GT-70, 1988.

13. Frink, N.T., "Recent Progress Toward a Three-Dimensional Unstructured Navier-Stokes Flow Solver," AIAA-94-0061, Jan. 1992.

14. Mongia, H.C., "Application of CFD in Combustor Design Technology," AGARD CP-536, Fuel and Combustion Technology for Advance Aircraft Technology, 1993.

15. Sturgess G.J., et. al., "Design and development of a research combustor for lean blowout studies," ASME Paper 90-GT-143, 1990.

16. Lee, J., "An Application of Two-Equation Model of Turbulence to Three-Dimensional Chemically Reacting Flows," AIAA-95-0734, January 1995.

17. Chen K.H. and Shuen J.S., "Three-Dimensional Coupled Implicit Methods for Spray Combustion Flows at All Speeds," AIAA-94-3047, June 1994.

18. Hsu A.T., Tsai, Y-L, and Raju, M.S., "A PDF Approach for Compressible Turbulent Reacting Flows," AIAA-93-0087, January 1993.

19. J.R. DeBonis, "Full Navier-Stokes Analysis of a Two-Dimensional Mixer/Ejector Nozzle for Noise Suppression," AIAA-92-3570, July 1992.

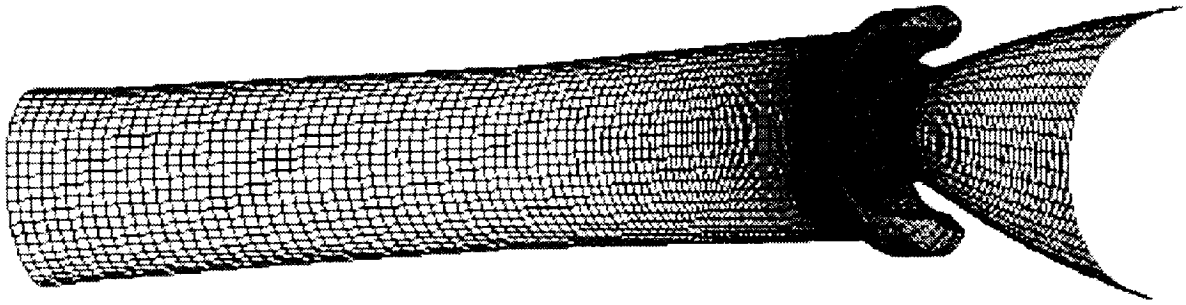
N95-28733

19951223/2

# MARSHALL SPACE FLIGHT CENTER SURFACE MODELING AND GRID GENERATION APPLICATIONS

Robert W. Williams, Theodore G. Benjamin, and Joni W. Cornelison
NASA George C. Marshall Space Flight Center
Computational Fluid Dynamics Branch
Marshall Space Flight Center, Alabama

## Gimballed Nozzles in Solid Rocket Motors



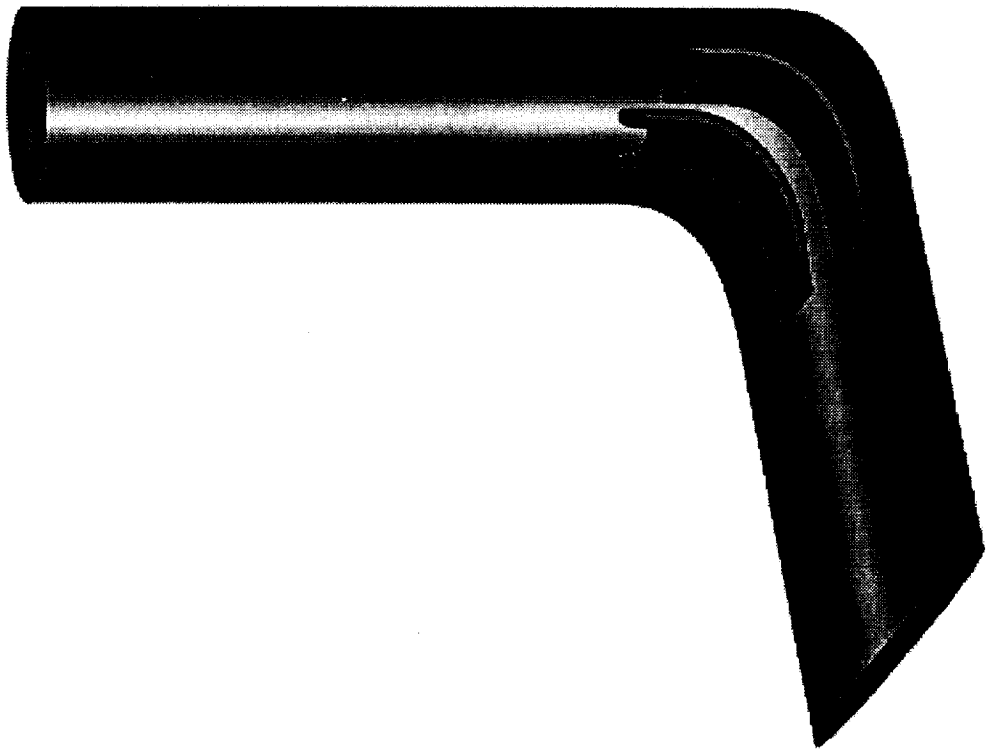
The Solid Rocket Motors (SRM) used by NASA to propel the Space Shuttle employ gimballing nozzles as a means for vehicular guidance during launch and ascent. Gimballing a nozzle renders the pressure field of the exhaust gases nonaxisymmetric. This has two effects: (1) it exerts a torque and side load on the nozzle; (2) the exhaust gases flow circumferentially in the aft-dome region, thermally loading the flexible boot, case-to-nozzle joint, and casing insulation. The use of CFD models to simulate such flows is imperative in order to assess SRM design.

The grids for these problems were constructed by obtaining information from drawings and tabulated coordinates. 2D axisymmetric grids were designed and generated using the EZ-Surf and GEN2D surface and grid generation codes. These 2D grids were solved using codes such as FDNS, GASP, and MINT. These axisymmetric grids were rotated around the center-line to form 3D non-gimballed grids. These were then gimballed around the pivot point and the gaps or overlaps resurfaced to obtain the final domains, which contained approximately 366,000 grid points. The 2D solutions were then rotated and manipulated as appropriate for geometry and used as initial guesses in the final solution. The analyses were used in answering questions about flight criteria.

For more information, contact Dr. Edward J. Reske.

PRECEDING PAGE BLANK NOT FILMED

## ALS LOX Sustainer Feed Line - Eighty-Degree Bend with Turning Vanes



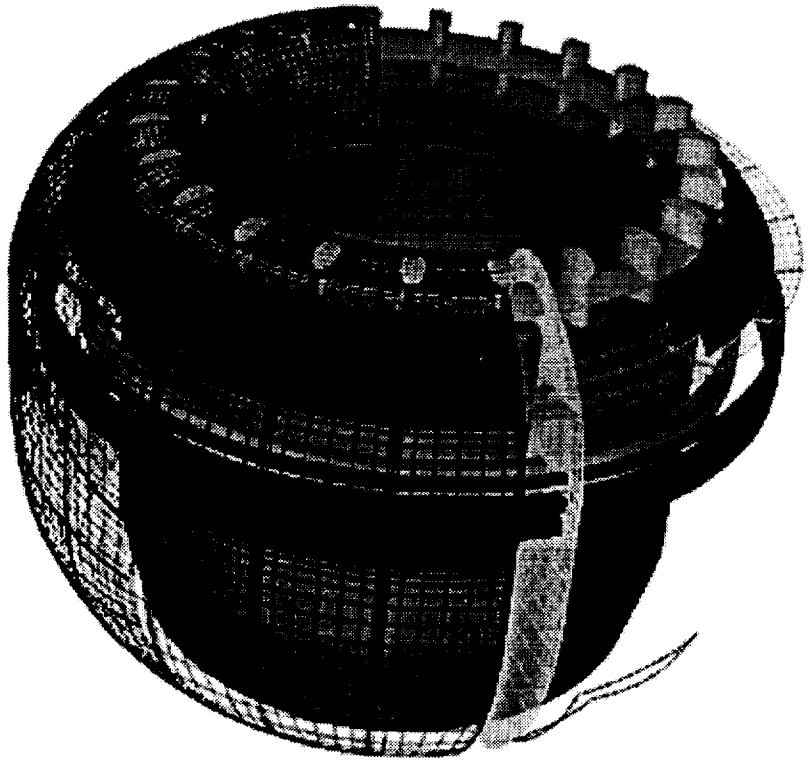
The purpose of this CFD study was to determine the vane loading and the amount of secondary flow downstream of an eighty-degree vaned pipe bend. The pipe in question was a liquid oxygen rocket engine feed line in the Advanced Launch System Propulsion module.

The geometry was generated on a Intergraph CAD system and transferred electronically by General Dynamics in NIGES format. The NIGES file was read into CAGI. Options were selected in CAGI to automatically discard many intermediate construction surfaces. The necessary surfaces were then extracted and written out in IGES format. The strong curvature of the guide vanes made it necessary to use a NURBS definition to properly model and maintain the blade shape. The IGES file was then read into GENIE++ where points were distributed on the surface and the volume constructed. Two grids were generated. Each grid had five zones totaling 115,000 and 360,000 grid points, respectively.

Since the version of INS3D that was used to solve the flow field did not support overset grids, an H-type grid was used to describe the geometry. This increased the complexity of adding the vanes. More recent releases of INS3D-UP have overset capabilities allowing greater grid generation flexibility.

For more information, contact Robert W. Williams.

## Alternate Turbopump Development Turnaround Duct

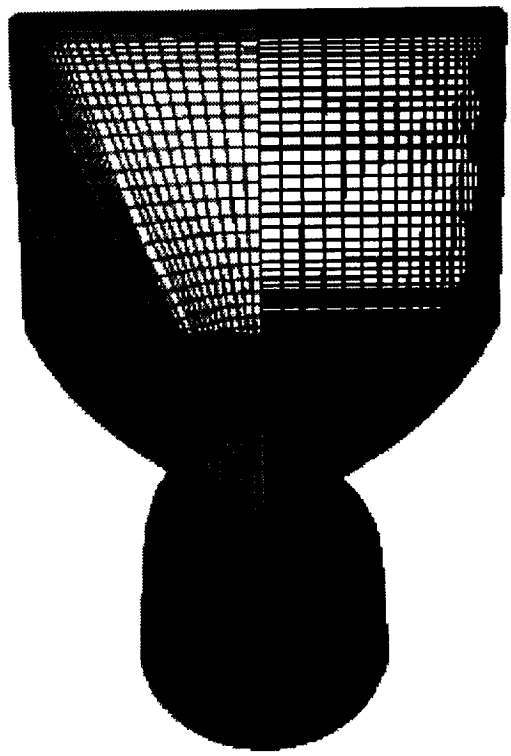


During testing of the ATD high pressure oxidizer turbopump at Stennis Space Flight Center cracks were found on the heat exchanger inner guide vane. A team was formed to assess the potential causes for the cracking and to propose changes to eliminate the problem. There was significant CFD analysis performed in support of this team. Over 50 2D or axisymmetric cases and six 3D cases were run. The actual turn-around duct geometry is 3D, but the 2D cases were run as axisymmetric to allow for rapid analysis of different configurations.

The geometries for the 2D grids were obtained partially from CAD systems and partially from faxed drawings. The grids were generated in GENIE++ and then smoothed using GEN2D. The 2D grids model a cross-section of the turn-around duct which contains a splitter vane, two heat exchanger vanes and heat exchanger coils. The 3D grid modeled the 22 struts on which the splitter vane is mounted. The data for the struts was obtained in the form of tabulated coordinates at midspan. Due to the nature of the 2D grids and to the time in which the grid was generated, the four intersecting boundaries along the strut were hand calculated. Later consideration indicates it may have been possible to use the surface-surface intersection option in GENIE++. A single passage between two struts was first generated and then rotated to obtain the full 3D grid. The 3D grid contains approximately one-half million grid points. REFLEQS was used to obtain the flow field for the 2D and 3D cases.

For more information, contact Joni W. Cornelison.

## Modular Thruster for a Tri-Propellant Engine

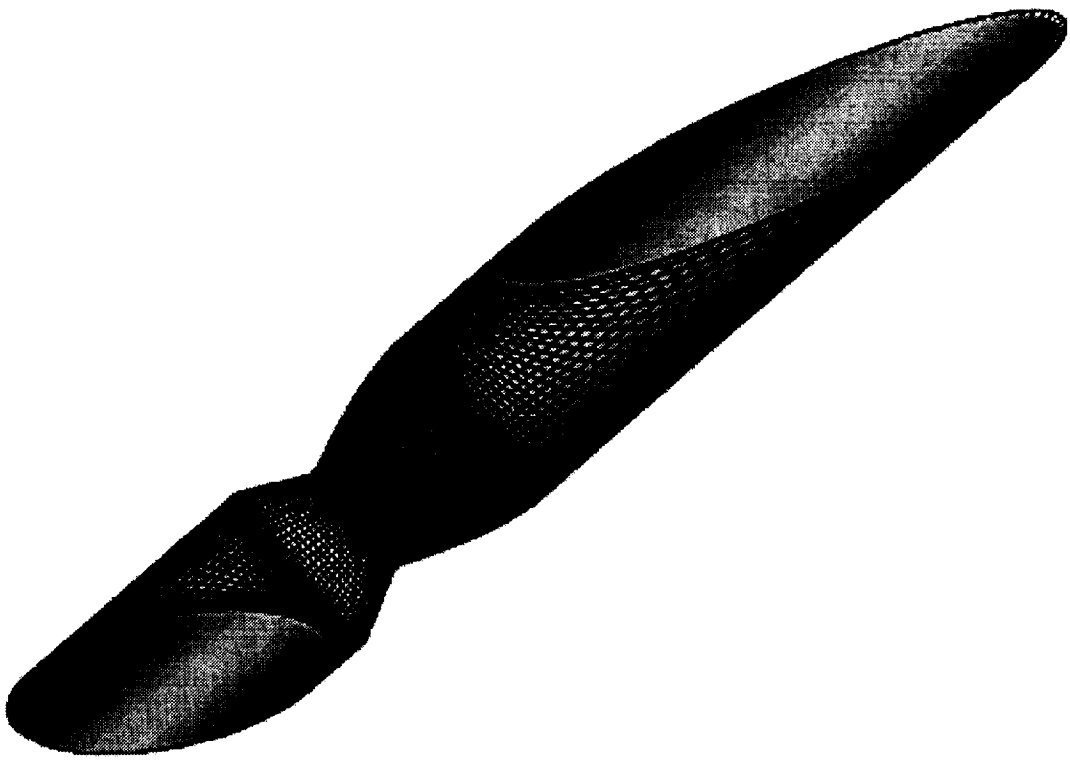


This geometry was received from Rocketdyne via an IGES file. The inlet region to the throat was a single NURBS surface; the exit from the throat was represented by two NURBS surfaces. All three surfaces were joined in NGP so that they could be split uniformly. Because there was no line of constant parametric value at the desired coordinates for splitting, the surface was split short of the desired coordinates and joined with a NURBS curve which lay on the symmetry plane. This was done in GENIE++ and these were combined on a NURBS surface. From that point in GENIE++, the flow field was discretized into an 87,451-point grid.

The flowfield was initialized using isentropic relationships and was solved in FDNS. This solution was checked for reasonableness and found acceptable. The purpose for this exercise was to develop a template for other similar geometries that are now beginning to be analyzed in tri-propellant engine design.

For more information, contact Theodore G. Benjamin.

## Simplex Turbopump Nozzle



Turbine aerodynamic design support was provided for the Simplex turbopump, an in-house project at Marshall Space Flight Center. The turbine was designed as a supersonic/impulse turbine. Several 3D CFD calculations of the nozzle were performed using the FDNS code. Results from the analyses were provided to support design changes and rotordynamics, dynamics, stress and thermal analyses.

The single-zone nozzle grid was generated entirely in GENIE++ and contains approximately 190,000 grid points. It was necessary to model the nozzle with a full 3D grid since the inlet plane and the exit plane are rotated 90 degrees apart. The cross-section of the nozzle was generated as an O-grid. Later consideration indicated an H-grid may have eliminated problems in smoothing the exit surface and an H-grid would have been necessary in generating a combined nozzle and turbine blade grid. The elliptic inlet and exit surfaces were obtained by generating elliptic boundaries and rotating to the proper angle. For this grid this approach was found to work better than using the surface-surface intersection option.

For more information, contact Joni W. Cornelison.

## Inducer Technology Model Volute Collector

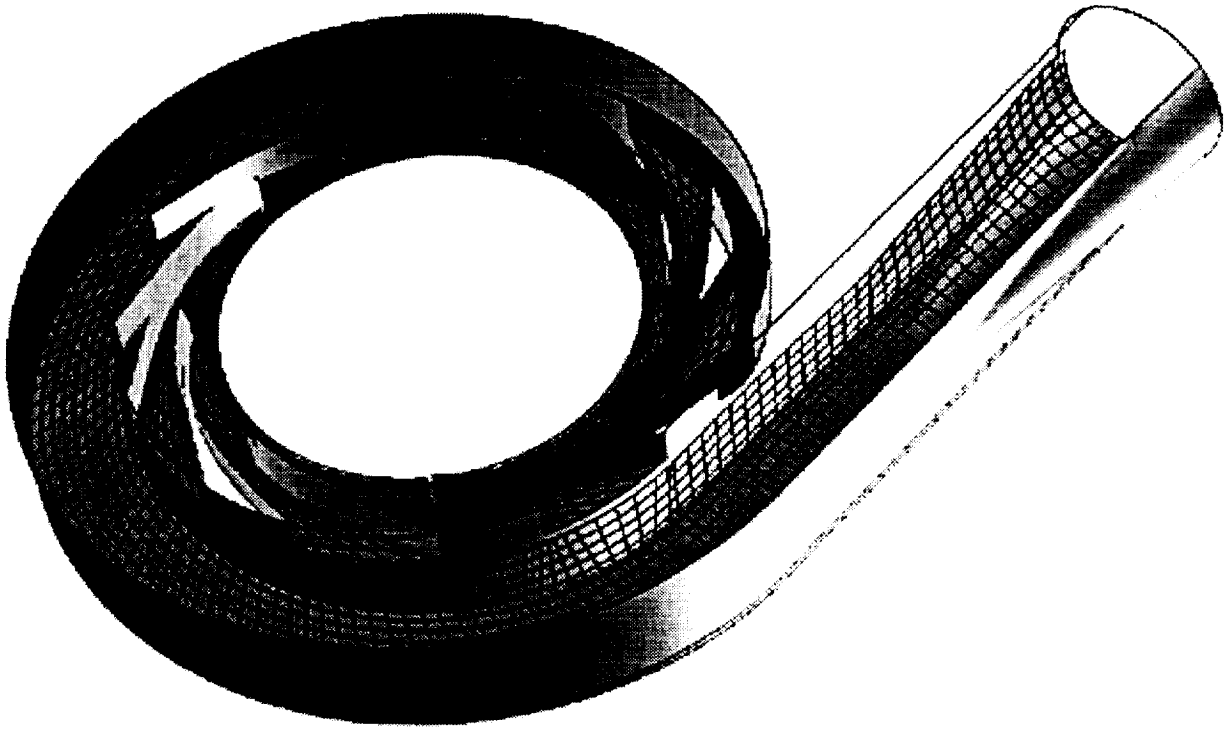
CFD analysis was performed on the volute in the ITM (Inducer Technology Model) to assess radial loads, total pressure losses, flow separations, and any unusual flow phenomena that might occur within the volute system.

The volute and exit pipe geometries were obtained from drawings. A 2D cross-section of the volute was quickly generated using GENIE++ and then input to GEN2D for smoothing. The 2D axisymmetric grid was read back into GENIE++ and rotated about the centerline to form the 3D grid. The grid which models the exit pipe connected to the volute was generated totally in GENIE++. The inlet section of the pipe is rectangular and intersects with a conic section near the exit of the pipe. The intersection of these two surfaces was performed successfully by using NURBS surfaces and the surface-surface intersection option. Various modifications were made to the volute housing and were quickly generated using GENIE++. The two-zone grid contains approximately 45,000 grid points.

FDNS was used to perform the incompressible flow analysis on three cases. Each case showed secondary flows in the volute housing and exit pipe. Based on these results, a honeycomb ring will be inserted at the pipe exit to eliminate the secondary flow.

For more information, contact Joni W. Cornelison.

## Simplex Diffuser and Volute

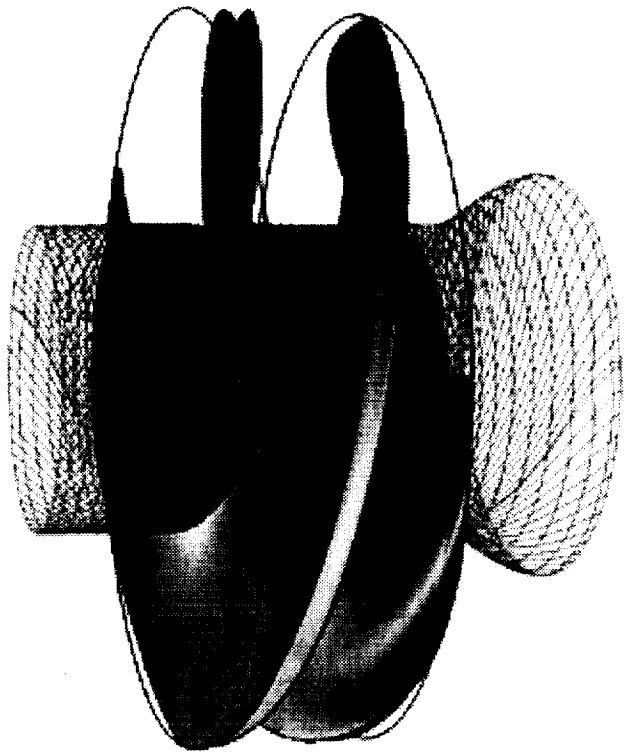


The Simplex Turbopump is an in-house project at Marshall Space Flight Center. The data for this geometry was extracted from drawings. The shape of the vanes is fully two-dimensional and it was modeled by fitting a curve through points which define the contour and stacking this contour axially. The volute is square in cross-section until the exit region, where the square cross section intersects with a conic section. This intersection was performed impressively well in GENIE++, which was utilized for the entire grid-generation process. The final grid contained 207,894 grid points in 10 zones.

This design was subjected to three-dimensional hydrodynamic analysis using the FDNS code. These analyses revealed a flaw in the initial design which caused separation on the pressure side of the vanes; this prevented the design from producing the desired diffusion. Alternative designs were proposed (and rapidly generated in GENIE++) and analyzed. As a result of the CFD analyses, one of these alternative vane proposals was incorporated into the final design before machining had begun.

For more information, contact Theodore G. Benjamin.

## Simplex Turbopump - Inducer



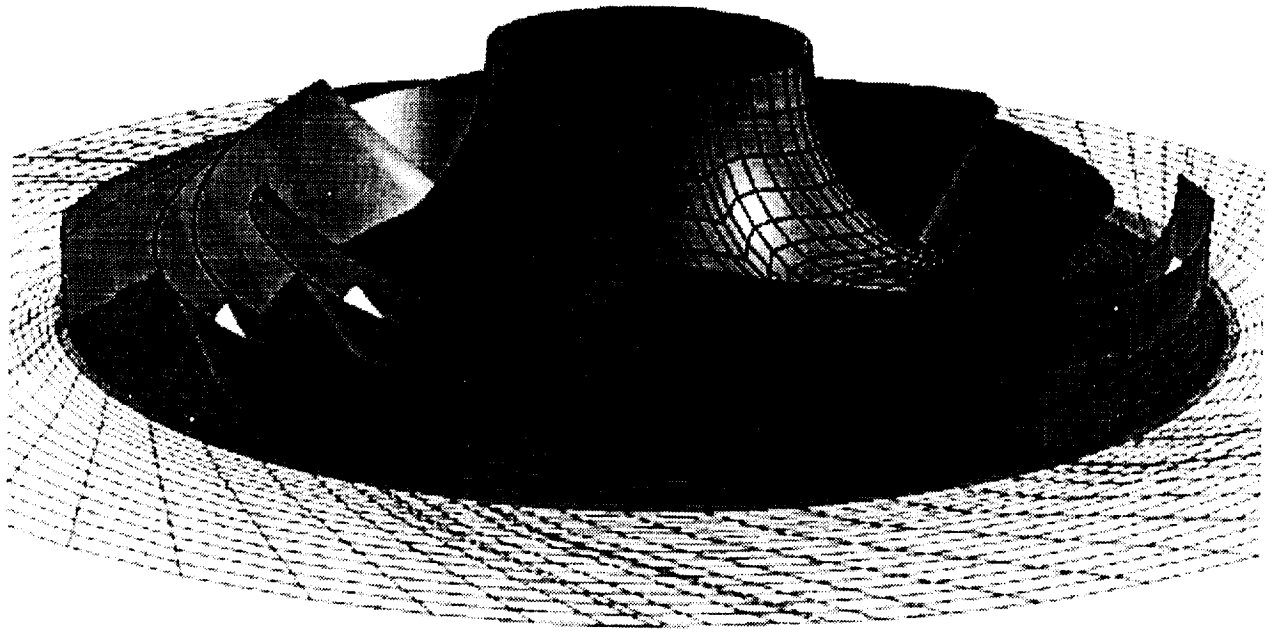
The Simplex Project is an in-house project at MSFC to design and build a turbopump to supply liquid oxygen to a hybrid rocket motor. The pump was required to deliver liquid oxygen at 570 gallons per minute and 1,500 pounds per square inch with a wheel speed of 25,000 revolutions per minute. The objective of this study was to use CFD during the design phase of the pump to predict performance and blade loading.

Blade surfaces were output from an Intergraph CAD system and transfered electronically in IGES file format. The surfaces described in NURBS were read into GENIE++ where points were distributed on the blades. The hub and shroud surfaces were generated by projecting an arbitrary surface constructed at the root and tip respectively on a boundary of revolution. For grid resolution studies, changes were made to appropriate indices in the GENIE++ history file and then the grid generator was rerun using those inputs. All grids consisted of three zones and ranged in size from 25,000 to 680,000 grid points.

Flow solutions were obtained using FDNS in the relative reference frame. Results were post processed using FAST and an in-house code to evaluate and make plots of performance and blade loads.

For more information, contact Robert W. Williams.

## Simplex Turbopump - Impeller



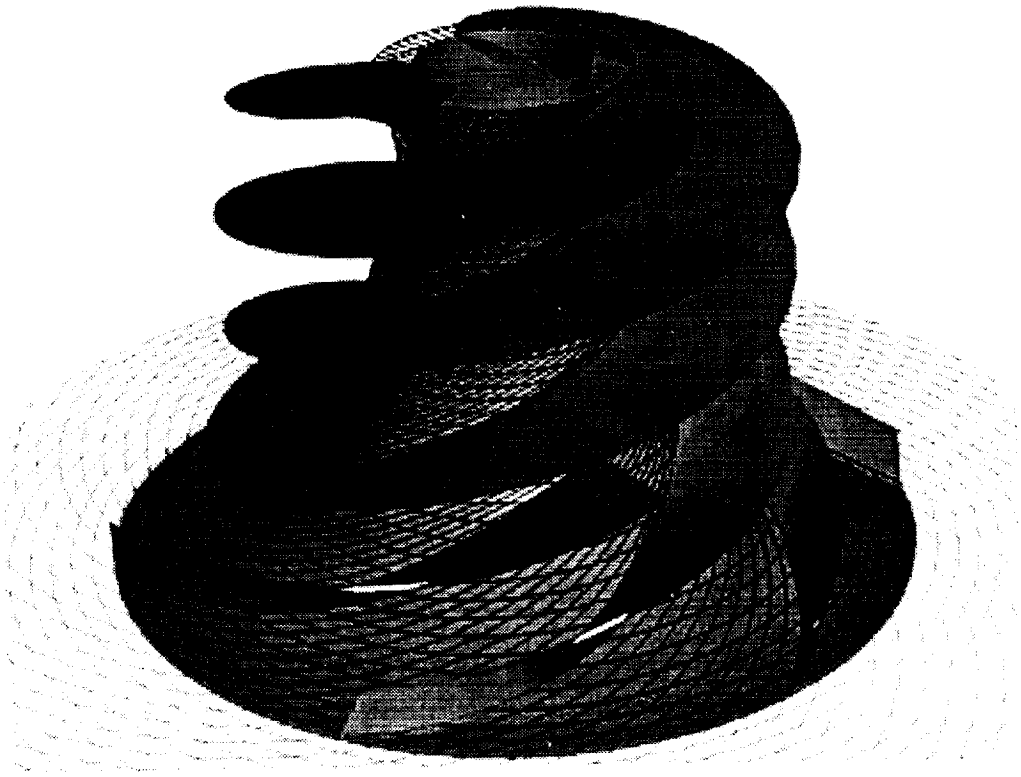
The Simplex Impeller was designed to operate in liquid oxygen at 25,000 revolutions per minute and deliver the fluid at 570 gallons per minute and 1,500 pounds per square inch. The purpose of this study was to aid in the design phase and determine performance and blade loading.

The geometry was transferred electronically from an Intergraph CAD system in IGES format. The NURBS surfaces were read into CAGI where streamlines were selected along periodic lines. The streamlines were set-up into an input file and read into TIGER. TIGER interactively generated the grid allowing control over Bezier curves connecting leading and trailing edges at the hub and tip. Several grids were generated in single and multiple zones. Grids ranged in size from 125,000 to 250,000 grid points. Later grids also modeled cavities between the impeller and diffuser. The cavities were added using GENIE++.

The flow field was solved using FDNS. Results were post-processed using FAST and an in-house code to calculate and plot performance and blade loading.

For more information, contact Robert W. Williams.

## Advanced Liquid-Hydrogen Turbopump - Inducer / Impeller with Single Splitter



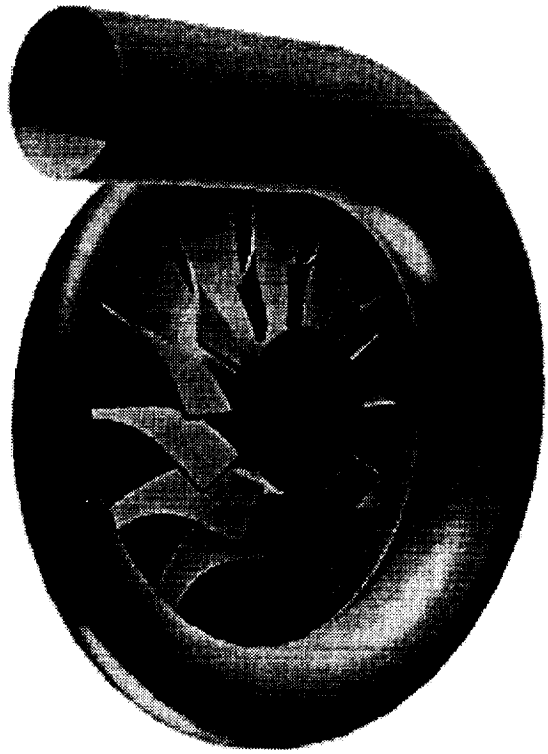
The development of the Advanced Long Life Turbopump is a cooperative agreement among the United States Air Force, NASA, and Pratt and Whitney. The pump was designed to deliver liquid hydrogen at 2,625 gallons per minute and has a rotational speed of 140,000 revolutions per minute. The pump is a combination inducer and impeller. In this analysis the pump was modeled with one set of splitter blades. It is shown above with three blades at the inlet and six blades at the exit. The actual geometry has three blades at the inlet and 24 blades at the exit.

Periodic conditions along each full blade were assumed simplifying the model to 120 degrees of the complete geometry. The geometry was received from P&W electronically in an ASCII text file. The geometry was represented by nine blade streamlines on each of the pressure and suction surfaces and the pump hub and shroud contours. The streamlines and contours were read into a translation program, the surfaces extracted, and written out into PLOT3D format. The surfaces were then read into GRIDGEN where points on the surface were redistributed and the volume was discritized. Two grids were generated. Each grid has four zones with the total number of grid points ranging from 16,000 to 116,000.

The problem was solved using FDNS in the relative reference frame at design flow conditions. Results were post-processed using FAST and an in-house code to calculate and plot performance and blade loading.

For more information, contact Robert W. Williams.

## Advanced Liquid-Hydrogen Turbopump Turbine and Volute



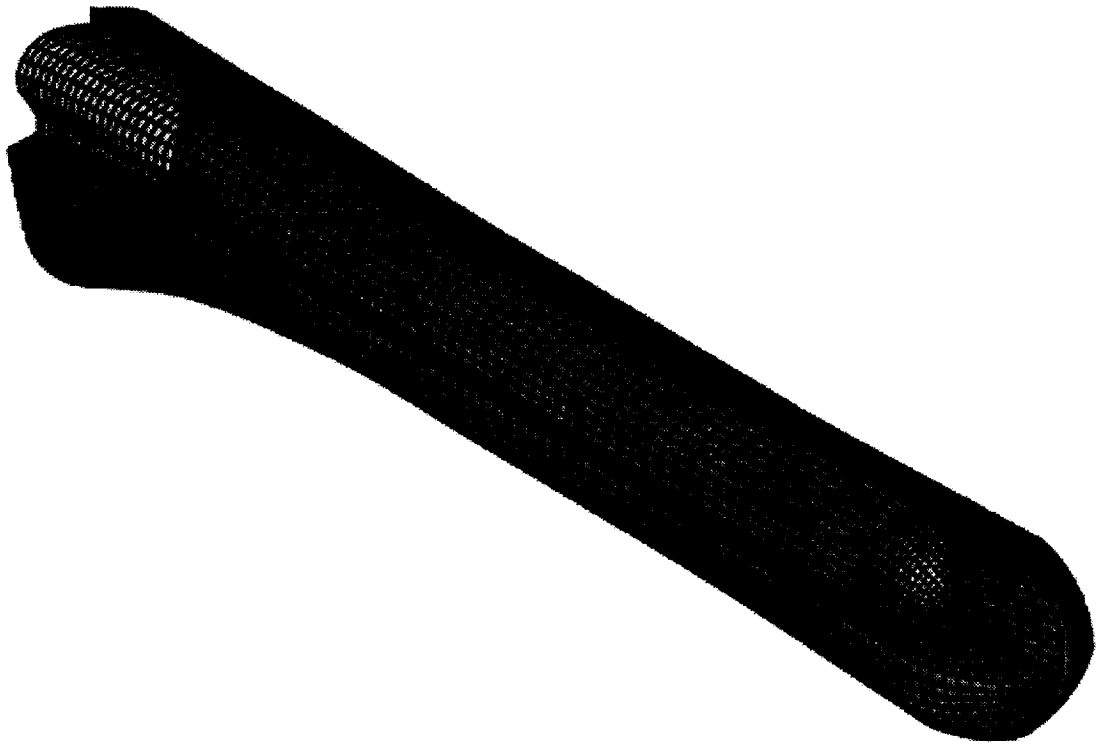
In support of the Air Force Phillips Lab ALT turbine aerodynamic design, NASA/MSFC is performing CFD parametrics to help determine optimum size and performance of this radial turbine. Unsteady CFD analysis is also being performed on the coupled volute and rotor. This turbine is in its design phase with the design work being performed by Pratt and Whitney.

The original surface grids for both sides of the rotor blade were obtained from an IGES file received from Pratt and Whitney. These surface grids along with the tabulated coordinates for the hub and shroud profiles were then input to TIGER. An initial volume grid was generated for a single blade passage and then input to GENIE++ to define the blade tip clearance region. The rotor grid currently contains three zones and consists of approximately 161,000 grid points. The volute grid is still in progress and when finished will be combined with a full 3D rotor grid.

Baseline balde analysis was performed using FDNS. The results indicate there are several problem ares that must be addressed.

For more information, contact Joni W. Cornelison.

## Simulation of Bone Cement Flow During Hip Implant Insertion



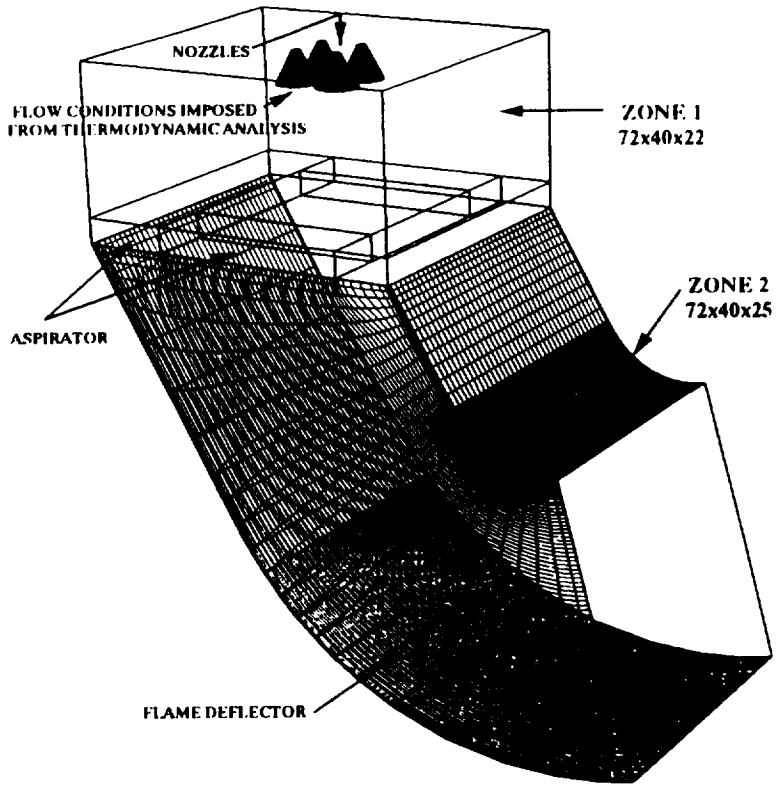
The purpose of this study was to simulate the flow of bone cement during hip implant insertion. For proper simulation of the induced cement flow, the surface of the stem and bone must move relative to each other at a velocity which corresponds to the insertion angle and rate. As a consequence, the grid must be continuously regenerated as the solution progresses. The mass flow of cement displaced out of the bone cavity is equal to the sum of the jacobian at each individual grid point.

The geometry definition was supplied by Howmedica Incorporated in IGES file format. The IGES file was read into CAGI where the hip-implant stem surface and the bone reamed-cavity surface were extracted and output in PLOT3D format. The two surfaces were then read into GENIE++ and transformed into NURBS surfaces for point distribution before building the discretized volume. The flow field for the 45,000 grid-point single-zone grid was then solved using FDNS modified to accommodate moving boundaries.

The analysis revealed the detrimental flow patterns which caused flaws in the solidified cement. The design was changed so that nonuniformity in the cement matrix was eliminated. A patent has been issued for this redesign.

For more information, contact Francisco Canabal.

## Pollutant Environment from RD-170 Propulsion System Testing

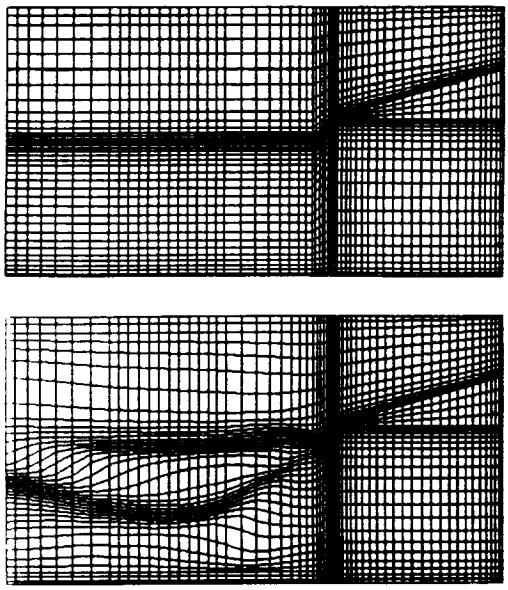


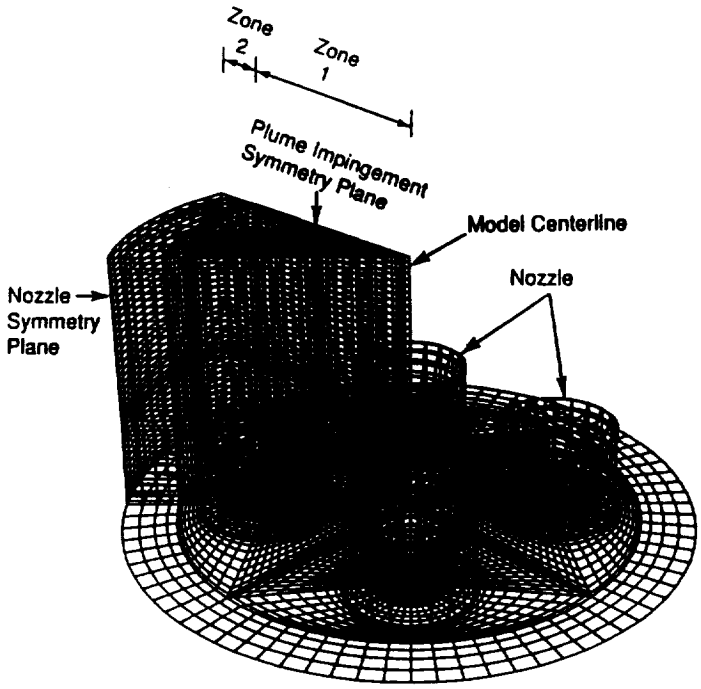

The objective of this study was to assess the exhaust-plume pollutant environment of the RD-170 engine during a hot-fire test on the F1 Test Stand at MSFC. A 3D simulation of the Russian-built kerosene engine was performed including: 3D air entrainment, 3D multiple-nozzle plume interaction and mixing with air, finite-rate after-burning reaction, plume impingement with flame bucket and plume quenching through deluge water, and 3D restricted plume expansion. Afterburning was modeled using 11-species, 18-reaction finite-rate chemistry. Water-quenching was computed assuming a homogeneous two-phase formulation.

Geometry was obtained from blueprints and test stand observation. Computational grid generation was performed using the EZSURF code. The edge curves of the nozzle exits, aspirator, flame deflector, and multi-zone block edges were generated first. Transfinite interpolation was then applied to create the initial surfaces. The flame deflector and nozzle exit surfaces were elliptically smoothed using Bezier curve and local redistribution techniques. The volume grid for the first block was created using two linear stackings -- one from the top of the block to the nozzle exit plane and then another from the nozzle exit plane to the bottom of the aspirator. The flame deflector block and subsequent atmosphere block volume grids were created using transfinite interpolation. The three-zone grid has a total of 300,000 points.

For more information, contact Dr. Ten-See Wang.

## Adaptive Gridding of a Four-Engine Clustered Nozzle Configuration





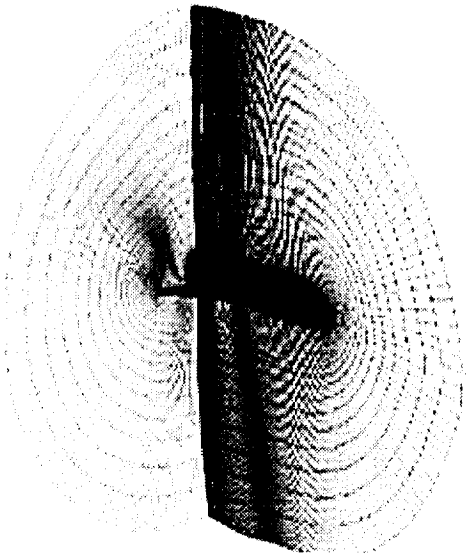

The objective of this study was to propose a computational methodology that could effectively anchor the base flowfield of a four-engine clustered nozzle configuration. For an efficient CFD calculation, a Prandtl-Meyer solution treatment was applied to the algebraic grid lines for resolution of initial plume expansion. As the solution evolved, the computational grid was adapted to the pertinent flow gradients.

Making use of the symmetrical nature of the geometry, a grid was generated for a small wedge of the complete flowfield. The "plume impingement symmetry plane" and the "nozzle symmetry plane" intersect along the "model centerline" at a 45-degree angle. Plume impingement and recompression would occur along the first symmetry plane and the centerline of the nozzle passes through the second symmetry plane. A two-zone baseline algebraic grid was generated with GENIE++. The first zone started from the base proceeding downstream, including the nozzle and plume region. The second zone (the outer shell) consisted of the ambient environment and a portion of the expanded plume. Grids ranged in size from 120,000 to 250,000 grid points. Initial plume-angle grid resolution was essential to the accurate prediction of base flow properties. Algebraic grids were generated for each pressure ratio resolving the initial plume angle by applying the isentropic Prandtl-Meyer plume-expansion theory.

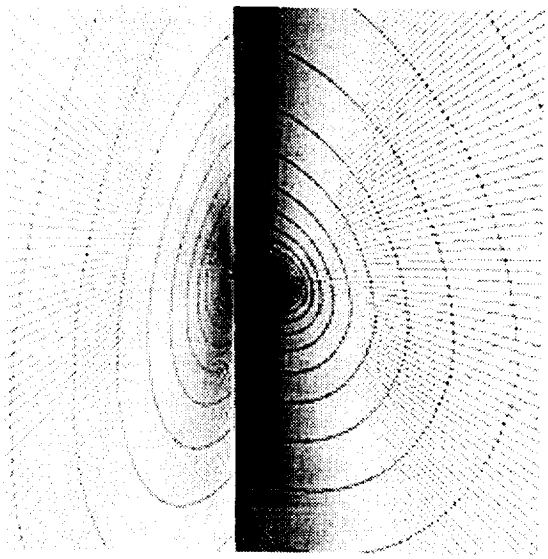
The SAGE code was used to refine the initial plume angle resolved algebraic computational grid. Since mach number contour was closely associated with the plume boundary layer and the pressure gradient follows the recompression shock, these two flowfield variables were used as pertinent grid-adaptation parameters. Adaptation was applied downstream of the nozzle lip to maintain the initial plume-expansion angle. Grid-line clustering follows both the plume boundary layer and the recompression shock.

For more information, contact Dr. Ten-See Wang.

## WB001 Wing-Body Vehicles



**The volume grid for supersonic flow simulations (159x129x43)**



**The volume grid for transonic flow simulations (159x129x65)**

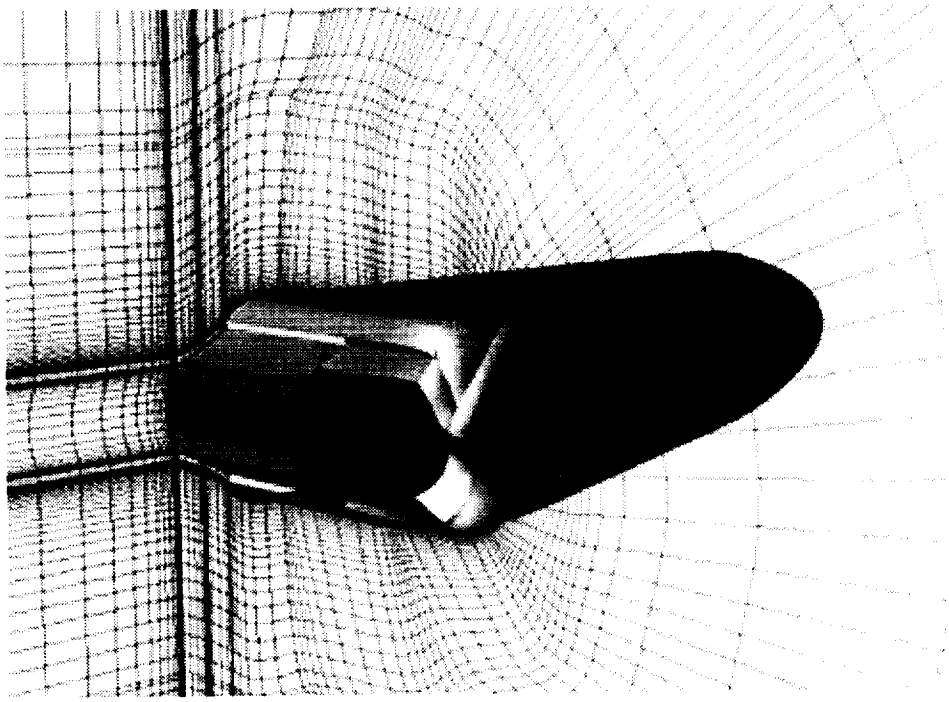
The original surface grid for the WB001was obtained from NASA-LaRC/MS66. This single-block grid contained 161x129 points. The volume grids were generated using HYPGEN. Since HYPGEN is a hyperbolic grid generator, there was no need to define the far field surface. Also, in order to obtain a smoothly stretched grid and sometimes to avoid negative volume grids, it was required to have a viscous spacing off the body. Subsequently, some layers near the surface could be removed for inviscid calculations.

The computational domain for each of the transonic and supersonic cases had to be properly generated to fully capture the physical phenomena. The figure on the left shows the volume grid generated for supersonic simulations. This grid was used for high supersonic simulation ($M_\infty$=5.72); therefore, the far field considered was close to the body, about 1 body length. This helped reduce the number of grid points used for the computation, thus less CPU time was required for the execution. Likewise, for transonic simulations ($M_\infty$=1.1) the far field had to be extended as far as possible to ensure free stream conditions. To be conservative, a far field of 10 body lengths was generated. The figure on the right shows the volume grid generated for transonic simulations.

It is noticed that the number of grid points in the k-direction was increased for the transonic simulations in order to retain a reasonable stretching ratio. Also, the number of grid points in the i-direction was changed so that mesh sequencing could be performed during flow computations.

For more information, contact Bruce T. Vu.

## Lifting Body with Aerospike Engines



The original surface geometry was provided as 18 coarse surfaces defining the body, wings, aerospike engines and body flaps. These surfaces were manipulated using a NURBS surface tool. The GENIE++ general purpose grid generation system was used to generate the volume grids. All the important features, such as fins, body flaps, nozzles, and base regions, have been included in this configuration. These grids were designed to aid in calculating plume aerodynamic interactions at subsonic conditions. For supersonic simulations the grid lines at the base regions could be modified to account for the plume angles. These structured grids contain 6 zones, totaling more than one-half million grid points, to be used for inviscid calculations.

For more information, contact Bruce T. Vu.

# STRUCTURED PATCHED GRID TECHNOLOGY

N95-28734

19951223 13

# AUTOMATIC BLOCKING FOR COMPLEX THREE-DIMENSIONAL CONFIGURATIONS

John F. Dannenhoffer, III*

United Technologies Research Center
East Hartford, CT 06108

## ABSTRACT

A new blocking technique for complex three-dimensional configurations is described. This new technique is based upon the concept of an abstraction, or squared-up representation, of the configuration and the associated grid. By allowing the user to describe blocking requirements in natural terms (such as "wrap a grid around this leading edge" or "make all grid lines emanating from this wall orthogonal to it"), users can *quickly* generate complex grids around complex configurations, while still maintaining a high level of control *where desired*. An added advantage of the abstraction concept is that once a blocking is defined for a class of configurations, it can be automatically applied to other configurations of the same class, making the new technique particularly well suited for the parametric variations which typically occur during design processes. Grids have been generated for a variety of real-world, two- and three-dimensional configurations. In all cases, the time required to generate the grid, given just an electronic form of the configuration, was at most a few days. Hence with this new technique, the generation of a block-structured grid is only slightly more expensive than the generation of an unstructured grid for the same configuration.

## BACKGROUND

One of the major impediments to the use of computational simulations for predicting the performance of real-world configurations is the time needed to set up a suitable computational grid. It is not uncommon for it to take well in excess of three months to generate a suitable grid for a new type of configuration. Clearly, such a long lead time makes the use of simulations practical for only a small number of important problems and for routine design work which involves relatively small changes from previous configurations (wherein the grid generation work can be amortized over many thousands of computational simulations).

Computational grids for fluid flows are especially hard to generate because of the stringent requirements imposed by the flow physics and subsequently the flow solvers (such as surface orthogonality and cell aspect-ratios of about $10^6$). It has proved to be very difficult to adapt the automatic techniques used within the structures community to situations with such stringent requirements.

In order to understand why grid generation is such a bottleneck, it is useful to take a quick look back at how block-structured grid generation software has evolved to its present state.

The first generation of block-structured grid generators were batch-oriented systems which extended, in a rather straightforward way, the grid generation techniques which had previously been developed for

*Senior Research Engineer

PRECEDING PAGE BLANK NOT FILMED

two dimensions. They allowed highly-trained experts to generate grids over arbitrarily-complex three-dimensional configurations through many thousands of lines of command-line control information. The most widely known of the first generation systems was **EAGLE**[1].

The second generation systems (which represent the current state-of-the-art) are workstation-based techniques which rely on high-speed graphics to allow users to interactively generate the same control information. With these systems, the experts can generate grids in less time, and with fewer errors, than they could with the batch-oriented systems. But the grids are still manually generated piece by piece, over a period of many months. Example second-generation systems include **NGP**[2], **ICEM-CFD**[3], **GRIDGEN**[4], **RAMBO-4G**[5], **3DGRAPE**[6], **PATRAN**[7], and **FIDAP**[8]. All of these suffer the same important disadvantage, namely that the grid generation process is very labor intensive.

In order to determine where all the labor is consumed, it is useful to break the process into its major steps:

**Prepare the configuration** — as expected, the first step in the grid generation process is the preparation of the configuration to be analyzed. The first part of this step requires that the grid generation system acquire the geometry, most preferably by a direct link to the CAD system on which the configuration was designed. Unfortunately, the configuration which the CAD system describes is seldom that which one wishes to analyze; for example, the actual configuration generally contains many geometric details (such as a probe) which are not germane to the analysis being performed. In addition, one often needs to create "artificial" surfaces to bound the domain, such as in the far-field or over an engine inlet. This preparation step usually accounts for the largest expenditure of labor. (It should be noted that this step is exactly the same, regardless of whether a block-structured or an unstructured grid is to be generated.)

**Design blocking plan** — the user has to decide upon a suitable grid topology for the given configuration, keeping in mind both the expected flow physics as well as any restrictions which the flow solver may impose. The user typically meets these requirements by choosing near-body topologies which are locally consistent with each component of the configuration, and then connecting these blocks in a way that ensures, or at least encourages, grid line smoothness in the field. At present, this step is an art, requiring much intuition, practice, patience, and unfortunately trial-and-error. This step does not in itself require much labor, but any changes made here can result in large labor expenditures in the following steps.

**Implement blocking plan** — the blocking plan just designed has to be described to the computer so that the grid can be generated. Current approaches require that a user transform the design (which the User's Manuals typically suggest that the user "sketch") into numbered or named points, lines, and surfaces. In addition the user must prescribe how the primitive components are to be connected in the final grid. The user is also required to set up the number of grid lines in each direction in each block. This step can consume a large portion of the block-structured grid-generation effort.

**Generate grid** — the grid point locations for all surface and field points are defined, either algebraically and/or through a PDE-based grid generator. This is the part of the process at which most current multi-block grid generators excel. In many systems, this step is a push-button operation which requires very little human time (although it may require a rather large amount of CPU time).

**Assess grid quality** — the final computational grid is examined for irregularities such as excessive skew and stretch, cells which are too small or large, etc. If the analysis of the grid reveals any deficiencies, then the user must go back to one of the steps listed above, fix errors or make different choices, and repeat. Although tedious, this step is usually performed rather quickly; the major impact of this step to the overall effort is the time incurred in iterating back to one of the steps above.

Currently, the major bottleneck is in the design and implementation of the blocking plan, steps 2 and 3 above. For complex three-dimensional configurations, these two steps can consume several man-months of effort. The objective of this work is to develop tools and techniques for efficiently performing these two operations for arbitrary three-dimensional configurations.

## TECHNICAL APPROACH

This paper presents a new method for designing, describing, and subsequently realizing, block-structured grids around complex three-dimensional configurations. It is assumed that the **Prepare the configuration** step described above has already been completed. Additionally, very little will be discussed about the **Generate grid** and **Assess grid quality** steps; both are discussed in detail elsewhere[9, 10].

### Description vs. Prescription

The basic idea behind the automatic generation of block structures is to change the user paradigm from one which is *prescriptive* to one which is *descriptive.*

In a prescriptive process, the user has a set of low-level tools which prescribe exactly *how* the problem is to be solved. Examples of these low-level tools include drawing a line in space to serve as an edge of some block, generating an edge which is constrained to lie along a certain input surface, and connecting four edges into a face. While control at this level is certainly very powerful (and sometimes indispensable), the labor associated with such an approach grows very rapidly, especially in configurations with multiple interacting components.

Alternatively, a descriptive process describes *what* the grid should look like. Example descriptions include the fact that the grid should wrap around the wing leading edge, that the store grid should lie within the wing grid, and that the grid should be orthogonal to all solid surfaces with a user-defined off-body spacing to the first field point. Such descriptive statements are very powerful; in fact, they generally correspond to between ten and a hundred low-level prescriptive commands. Additionally, since descriptive commands represent high-level concepts, they provide a framework for divide-and-conquer approaches, wherein the grid in the vicinity of each component is described separately and the automatic blocker worries about interfacing them to achieve an overall block-structured grid.

It should be noted that there is an exact equivalence between the prescriptive and descriptive approaches; the only real differences between the two is the economy of the expression which the descriptive approach provides.

### Abstraction

The key to describing, rather than prescribing, a block-structured grid is the direct preparation (by the user) of an abstract "sketch", herein called an *abstraction.* The abstraction can be thought of as a "squared-up" representation of the given configuration and of the computational grid which is to be produced. As will be seen below, typical configurations are generally abstracted by brick-shaped elements which represent the various components of a configuration.

This idea was inspired by the earlier work of Allwright and his colleagues[11], who showed the utility of such an abstraction in describing arbitrarily-complex three-dimensional grid topologies. Related work by Shaw and Weatherill[12] has also used the concept of abstraction. In both cases, the configurations of interest were all airplane-like, consisting of components such as fuselages, wings, and nacelles.

Although there are many similarities between the present work and the works of Allwright and Shaw, there are some major differences which make the present work more amenable to general configurations

such as are found in internal flows (for example, ducts and pumps) and traditionally non-aerodynamic configurations such as elevator hoistways. (Details of these differences will be published at a later date.)

The exact technique used to specify the abstraction of both the configuration and grid and their properties in the current technique is through the drawing of *blocking objects* on an abstract *background grid.* These blocking objects, which come in a variety of types (cubes, wraps, attaches, holds, rulers, and spacers), closely correspond to the concepts which one "sketches" during the design of a block-structured grid.

In the sections which follow, each type of blocking object, and the role it plays, is described. For illustrative purposes, the configuration which is used is a two-dimensional airfoil in a wind tunnel. It should be noted that the tools and techniques are equally applicable to more complex configurations in two and three dimensions, as demonstrated in the **Applications** sections below.

## The Background Grid

The space in which the abstraction is drawn is covered by an integer Cartesian grid called the *background grid*, which can be thought of as a piece of graph paper (in either two or three dimensions). The resolution of the background grid is chosen so as to be fine enough to describe the various components of the configuration and block structure, yet coarse enough to allow background grid operations to be performed efficiently. All blocking objects are placed on the background grid in such a way that their corners or endpoints lie exactly on an integer coordinate of the background grid (that is, where the lines on the graph paper cross). While this limits the number of locations and the sizes of the blocking objects which are possible, such a restriction is warranted since it makes it possible to efficiently determine the relative locations of various objects (such as object "A" is to the left of object "B" or that object "C" is contained within object "D"). Besides the ability to specify relative positions, the actual background grid coordinates at which an object is placed is arbitrary. That is to say, the absolute distance between points on the background grid is insignificant.

## Abstraction of Configuration: Bound Cubes

The first step in describing a grid is the creation of an abstraction of the configuration around, or inside, which a grid is required. This abstraction, or "squared-up" representation, is built up with a series of brick-shaped objects which are call *bound cubes.* In the current implementation, the user "draws" bound cubes by selecting a pair of background grid locations to specify opposite corners of the bound cube; the rectangular region between these points is automatically filled in with the bound cube which is rendered as a solid-colored box.

As an example, consider the configuration made up of an airfoil and its wake in a wind tunnel; a suitable abstraction is shown in Figure 1a. Notice that the airfoil, wake, and outer boundary are each represented as a rectangle (even though the actual geometry is not rectangular). Of significance in this figure is the fact that the airfoil is completely contained within the outer boundary, that the wake is attached to the right of the airfoil, and that the wake is also connected to the downstream (right) portion of the outer boundary. Recall that since the size of each increment on the background grid carries no significance, the airfoil bound could have been placed anywhere within the outer boundary, yielding a completely equivalent representation.

At this point, a slight digression is in order to describe the basic interpretation of the abstraction. The default block-structured grid which is generated corresponds to an H-type grid. Blocks are generated to fill the portions of the background grid which are outside the bound cubes. Specifically, block boundaries will be generated to yield the minimum set of blocks which are both all rectangular (in the abstract space) and which have one-to-one face matches with all other blocks; additionally, each face of each block can at

most be associated with one configuration surface.

As in all H-type grids, the grid lines basically run horizontally and vertically throughout the domain. So for Figure 1a, there are some grid lines which begin at an upstream portion of the outer boundary (in the region labeled "a") which run horizontally and which pass below the airfoil/wake and end at a downstream portion of the outer boundary (labeled "g"). Similarly, grid lines which begin in the region labeled "c" pass above the airfoil and end in the region labeled "h". Finally, there are grid lines which begin upstream (to the left of) of the airfoil (in the region labeled "b") which end at the front of the airfoil surface.

The other family of grid lines (those which run vertically on the background grid), has similar behaviors. Some start at the lower portion of the outer boundary (in the region labeled "d"), pass in front of the airfoil, and end at the upper portion of the outer boundary (labeled "i"). Others start in the regions labeled "e" and "f" and end on the lower surface of the airfoil and wake, respectively. Still others begin on the upper airfoil and wake surfaces and end on the upper boundary in the regions labeled "j" and "k", respectively. The corresponding grid for this configuration is shown in Figure 1b. Note that seven blocks are created, with block boundaries emanating only from the "corners" of the airfoil.

In the above discussions, the terms "lower surface of the airfoil", "upstream portion of the outer boundary", etc., should not be taken too literally; the terms simply serve to indicate approximate parts of the configuration, with the exact locations determined automatically so as to make the grid as smooth as possible. Note also that terms such as "some grid lines" were used in the above discussion; the exact number of grid lines in each region is specified by another blocking object known as a ruler (which is described below).

### Abstraction of Grid: Field Cubes and Wraps

As stated above, once the configuration has been abstracted with bound cubes, a default H-type grid can automatically be generated. In order to change the grid topology either locally or globally to another type, such a C-type, other blocking objects must be added to the background grid. Field cubes and wraps are those blocking objects.

A *field cube* is a rectangularly-shaped box which is placed on the background grid to force additional block boundaries to be generated. As a general rule, all grid blocks will lie either within or outside of each field cube. Notice that field cubes do not change the topology of the grid; they just give the user a means of breaking the default blocks into smaller blocks.

When the background grid is first generated, a special field cube called the outer boundary is placed around the periphery of the background grid. All other blocking objects must be placed on or within this outer boundary — a rule which is consistent with the notion of an outer boundary.

For example, consider again the airfoil/wake in a wind tunnel. If a field cube is added (as shown by the unfilled rectangle in Figure 2a), then the grid which is generated is still H-type (see Figure 2b), but now contains 18 blocks. In concept, breaking blocks in this way should not change the grid point locations; in practice, the locations are slightly different due to the fact that the elliptic smoother is not run until convergence.

So far, the bound and field cubes have been sufficient to describe "streamline-like" grids, but not "wrap-around" grids, as are frequently required. To remedy this, *wraps* are added to the background grid.

Wraps are specified by selecting two cubes (either bound or field), and then automatically creating "diagonal" lines to connect their respective corners. Shown in Figure 3 are the nine different wraps which are possible in two dimensions; in three dimensions, the number grows to twenty-seven. The first wrap shown in the Figure, in which the inner cube is fully within the outer cube, corresponds to an O-type grid: one family of grid lines are bent by each of the diagonal corners such that they encircle the inner cube while the other family are basically radial. The second row of wraps in the Figure, which correspond to

C-type regions, result when one edge of the inner cube is coincident with an edge of the outer cube. The last row corresponds to corner wraps.

Figure 4 shows the effect of wrapping the airfoil bound cube within the field cube which was added in Figure 2. This results in a C-type grid around the airfoil, but within an outer H-type grid.

If one considers the trapezoidal regions generated by a wrap to be field cubes, then the concept of wrapping can be nested to create an almost unlimited set of block topologies for a given configuration. For example, one may want to create a wrapped grid around a fuselage. Then in the region next to the fuselage which contains the wing, another wrap may be desired to create a C-type grid on the wing. Then, the region below the wing can be further wrapped around a pylon to create a local C-grid there. Although it may seem confusing, users have reported that thinking about the block structure in this manner is much less confusing than the alternative of manually constructing complex block structures for such configurations.

### Attachment between Abstraction and Configuration: Attaches

A new development of this work over that previously reported[13, 14] is the method by which the abstraction and configuration are attached. Presently, a very simple blocking object called an *attach* is used by which the user gives the correspondence between each corner of each bound cube (and the outer boundary) with the appropriate location on the configuration. For example, the attaches needed for the airfoil/wake configuration (discussed previously) are shown in Figure 5a. The easiest place to actually see the attaches are those associated with the airfoil and wake; the attaches at the corners of the outer boundary are not as obvious because the abstraction and configuration happen to be rendered at the same location, masking the line which connects the two. The attaches associated with the airfoil are of two types: those drawn as a star specify that the grid generator is required to put the grid point at the *exact* configuration point to which it is attached; those drawn as a circle specify that the grid generator can "slide" the grid point along the selected configuration surface so as to make the grid as smooth as possible. The latter is a particularly nice feature since it eliminates the need for the user to exact specify *a priori* all block corners which are on the interior of a surface.

This Figure also points out another feature of the abstraction. The mapping between the configuration and the abstraction is a many-to-one relationship. That is to say, each point on the abstraction corresponds to one point on the configuration, whereas each point on the configuration can correspond to many points in the abstraction. This is evident at the airfoil trailing edge, where the trailing edge point maps to both the top and bottom of the airfoil's abstraction.

### Other Blocking Objects: Rulers, Holds, and Spacers

In the above descriptions, terms such as "some grid lines" were used to describe the number of grid lines in each region of the grid. A blocking object called a *ruler* is the method for exactly specifying the number of points in each region. Shown in Figure 5b are the rulers for the H-grid problem.

One of the useful features of the abstraction is that it is relatively easy to propagate sizes from one region to another; grid regions (blocks) which abut are required to have the same number of grid lines. Similarly, it is relatively easy to perform size arithmetic, that is to determine the number of grid lines in various regions by simple arithmetic calculations. The only restriction on sizes is that they be consistent (that is, all size arithmetic yield the same results) and that enough sizes be prescribed so that the number of grid lines in all blocks can be calculated.

Another blocking object called a *hold* gives the user a way of easily specifying geometry such as symmetry planes and outer boundaries. A hold represents a plane in physical space in which one of the space coordinates is held fixed; the value of the fixed coordinate is determined automatically by the attaches which are associated with the hold. Unlike surface-surface intersections which must be explicitly generated,

the intersection between two holds or between a hold and a surface does not have to be directly generated by the user; this makes holds an easy way of representing symmetry planes for half-body configurations. Because holds act as a surrogate for an actual configuration surface within the grid generation process, they will not be discussed further here.

The final blocking object, a *spacer*, is the method used to describe grid properties such as wall spacings and orthogonality. Rather than requiring that the user specify these quantities along the edges which are (logically) perpendicular to a surface, spacers allow the user to specify the surface directly. Spacers give the user the ability of directly describing the grid requirement the "grid lines should be normal to the wing surface and that the wall spacing should be x." Since spacers do not directly affect the blocking, but rather are passed along to the grid generator in the form of required edge spacings, they will not be discussed further here.

### Transformation of Abstraction to Grid

Once the grid topology is specified by placing blocking objects on the background grid, a set of transformations, which are fully described in [13], are used to generate a suitable assembly of grid blocks. The key steps include:

- transform the field cubes and wraps into grid blocks;
- fill the remaining portion of the background grid with (logically-)rectangular grid blocks
- break grid blocks so as to establish one-to-one face matches (partial face matches are not allowed — see [15] for a method for overcoming this limitation);
- determine the sizes (number of grid points) in each block;
- eliminate degenerate blocks (that is, blocks which contain no grid points);
- associate the abstraction with the configuration surfaces and curves;
- allocate grid nodes for each block;
- use parametric transfinite interpolation to assign initial locations to all grid nodes which are constrained to lie on one (or more) of the configuration surfaces;
- use transfinite interpolation in space to assign initial locations to all other grid nodes; and
- apply a multi-block elliptic grid generation schemes to smooth the block boundaries (which were arbitrarily placed above).

## TWO-DIMENSIONAL APPLICATIONS

This section begins by showing the three different grids around an airfoil/wake in a wind tunnel which were previously described in the **Abstraction of Configuration: Bound Cubes** and **Abstraction of Grid: Field Cubes and Wraps** sections. Figures 1, 2, and 4 show the abstractions and resulting grids for two different H-type and one CH-type grid. In all cases, the same configuration abstraction (bound cubes) and abstraction-to-configuration attachments (attaches) were used; the only differences were in the grid abstractions (field cubes and wraps). In other words, once the time-consuming task of setting up a configuration was done, the new technique allowed the user to quickly experiment with differing grid

topologies. This collection of grids required less than five minutes to specify and generate on an engineering workstation.

To show the generality of this abstraction technique, grids for two completely different configurations were also generated. Figure 6 shows the abstraction and grid for an internal-flow passage containing eight circular posts and Figure 7 shows the abstraction and grid for a cross-sectional cut through a gas turbine combustor. Both grids required less than 30 minutes to describe and generate.

## THREE-DIMENSIONAL APPLICATIONS

The first demonstration presented here is the full three-dimensional model of a gas turbine combustor, as shown in Figure 8. This configuration was abstracted in steps. First, an abstraction from an equivalent two-dimensional configuration was generated and then extruded into the third dimension. Second, the hole associated with the fuel injector was cut into the abstraction, with the grid blocks automatically readjusting themselves. And third, the eight dilution holes (four each in the upper and lower liners), were added with the grid blocks again responding. This multi-step process significantly reduces the number of items which the user has to consider at any one time, and thereby helps limit the number of user errors.

The second three-dimensional configuration is for a generic elevator hoistway with a streamlined car (Figure 9). This configuration, which is very different from the above, shows that the current set of blocking objects is flexible enough to handle very diverse configurations. This grid was generated in less than a day.

The third configuration is a generic fighter with a side-mounted inlet (shown in Figure 10). This configuration, which is more typical of those found in the aerospace industry, exhibits complications in the area where the engine duct and the horizontal tail intersect. (This is the reason why the abstraction of the horizontal tail is fatter than the wing.) In generating the configuration abstraction for this case, the configuration was conceptually broken into a series of sub-problems, each of which were attacked independently. The solutions to these sub-problems were stored as templates which were recalled and combined to tackle this complex problem. The entire grid for this case, including the development of the templates, took less than 3 days to complete, given just an electronic specification of the configuration.

Finally, a grid was generated for another fighter which was topologically similar to that shown in Figure 10, but which had a wing with a different planform and airfoil section. This grid (not shown because it essentially looks the same as that in Figure 10) was generated *completely automatically* with the exact same blocking objects as were used above. This demonstrates another power of this technique — parametric grids can be generated automatically for classes of configurations once the first grid has been generated.

## CONCLUSIONS

A new technique which enables a user to describe, rather than prescribe, block-structured grids in two and three dimensions is presented. This new technique has many advantages over current techniques, namely:

- the blocking for a completely new configuration can be done in about an order-of-magnitude less time than is typically required using more traditional techniques. This makes block-structured grid generation again competitive with unstructured grid technology for a wide variety of problem types;
- since the configuration and/or grid topology can be easily modified, the user is encouraged to employ a divide-and-conquer approach to complex configurations. That is, the current technique allows users to build up configurations slowly, thereby reducing the complexity added at any step (and the associated chance of making an error);

- since a grid topology can be modified rather simply, the trial-and-error process which is unfortunately part of blocking design, is easily accommodated, with the end result being higher quality grids;
- templates of abstractions can be created for many frequently-occurring configuration components, thereby reducing the blocking problem for multi-component configurations to the concatenation and nesting of the appropriate templates. In fact, previous work in automatic grid generation[16] used optimization techniques and expert system technology to automatically generate grids in two dimensions; and
- the technique described herein is independent of the multi-block grid generator used, and thus is immediately applicable as a front-end to many current systems.

## ACKNOWLEDGEMENTS

This work was conducted under the United Technologies Corporate Sponsored Research Program and is currently implemented both as a stand-alone system (called **MKBLOX**) and as an application module within the National Grid Project (**NGP**). The author would like to thank Prof. Nigel Weatherill for the many hours of discussions about this work. Also, the author would like to thank his many colleagues who suffered through many demonstrations of half-baked ideas; their comments were invaluable in making this technique applicable to real-world problems.

## REFERENCES

[1] Thompson, J.F., Lijewski, L.E., and Gatlin, B., "Program EAGLE User's Manual," AFATL-TR-88-117 (Volumes I, II, and III), September, 1988.

[2] Gaither, A., Gaither, K., Jean, B., Remotigue, M., Soni, B., Thompson, J., Whitmire, J., Dannenhoffer, J., Weatherill, N., "The National Grid Project: A System Overview," NASA-CP-3291, May, 1995.

[3] Akdag, V., and Wulf, A., "Integrated Geometry and Grid Generation System for Complex Configurations," NASA-CP-3143, pp 161–171, April, 1992.

[4] Steinbrenner, J.P., Chawner, J.R., and Fouts C.L., "The GRIDGEN 3D Multiple Block Grid Generation System," WRDC-TR-90-3022 (Volume II), July, 1990.

[5] Widhopf, G.F., Visich, M., Than, P.T., Boyd, C.N., Huang, S-C., Oliphant, P.H., Stall, D.J., Swedberg, G.D., Shiba, J.K., and Coons, R.E., "Rambo-4G: An Interactive General 3-D Multi-Block Grid Generation and Graphics Package for Complex Multi-Body CFD Applications," Aerospace Report ATR-91(9990)-1, October, 1990.

[6] Sorenson, R.L., "The 3DGRAPE Book: Theory, Users' Manual, Examples," NASA-TM-102224, July, 1989.

[7] "PATRAN Plus User's Manual," PDA Engineering, July, 1987.

[8] "FIDAP Program" documentation, Fluid Dynamics International, Inc., January, 1990.

[9] Thompson, J.F., Warsi, Z.U.A., and Mastin, C.W., *Numerical Grid Generation: Foundations and Applications*, North-Holland, 1985.

[10] Parmley, K.L., Dannenhoffer, J.F., and Weatherill, N.P., "Techniques for the Visual Evaluation of Computational Grids," AIAA-93-3353, June, 1993.

[11] Allwright, S.E., "Techniques in Multiblock Domain Decomposition and Surface Grid Generation," in *Numerical Grid Generation in Computational Fluid Mechanics '88*, Pineridge Press Limited, 1988.

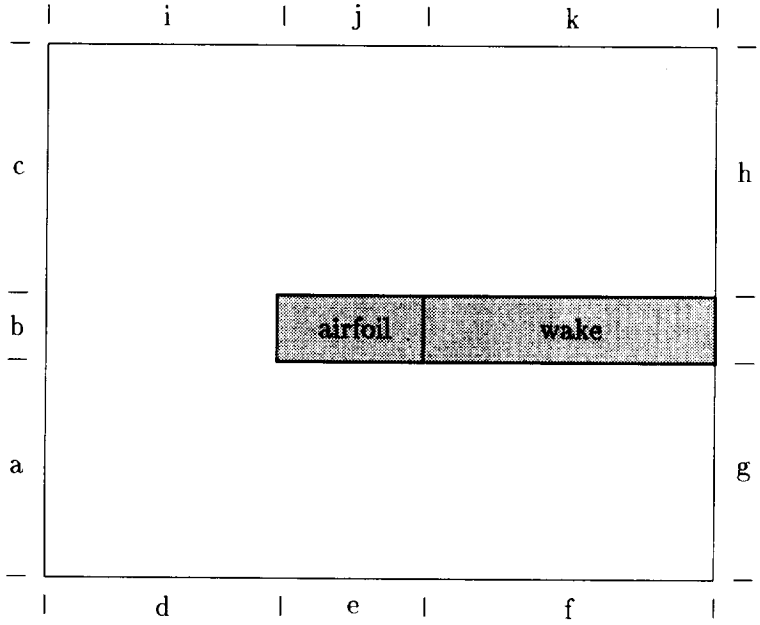
[12] Shaw, J.A., and Weatherill, N.P., "Automatic Topology Generation for Multiblock Grids," *Applied Math. and Comp.*, vol 53, pp 355–388, 1992.

[13] Dannenhoffer, J.F., "A Block-Structuring Technique for General Geometries," AIAA-91-0145, January, 1991.

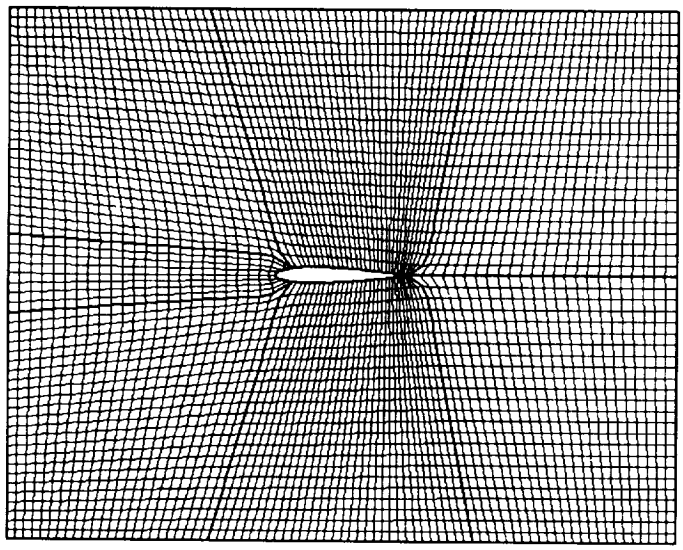
[14] Dannenhoffer, J.F., "A New Method for Creating Grid Abstractions for Complex Configurations," AIAA-93-0428, January, 1993.

[15] Dannenhoffer, J.F., "A Technique for Optimizing Grid Blocks," NASA-CP-3291, May, 1995.

[16] Dannenhoffer, J.F., "Computer-Aided Block-Structuring Through the Use of Optimization and Expert System Techniques," AIAA-91-1585, June, 1991.

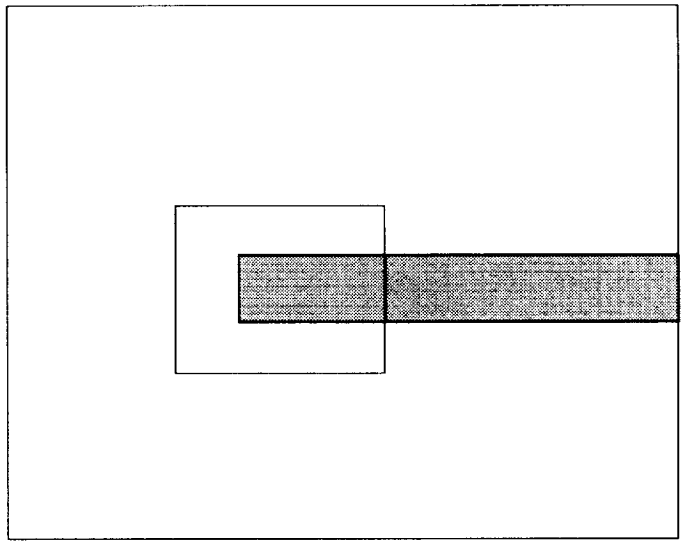



a) abstraction

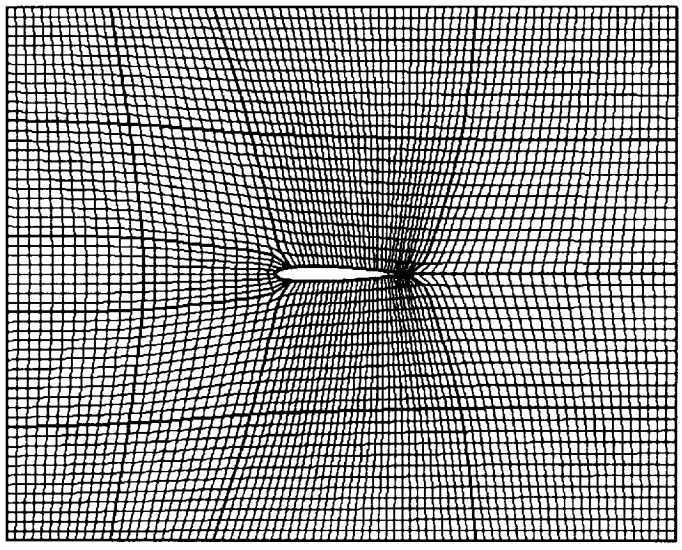


b) grid

Figure 1: H-grid around airfoil in wind tunnel.



a) abstraction



b) grid

Figure 2: H-grid (with field cube) around airfoil in wind tunnel.

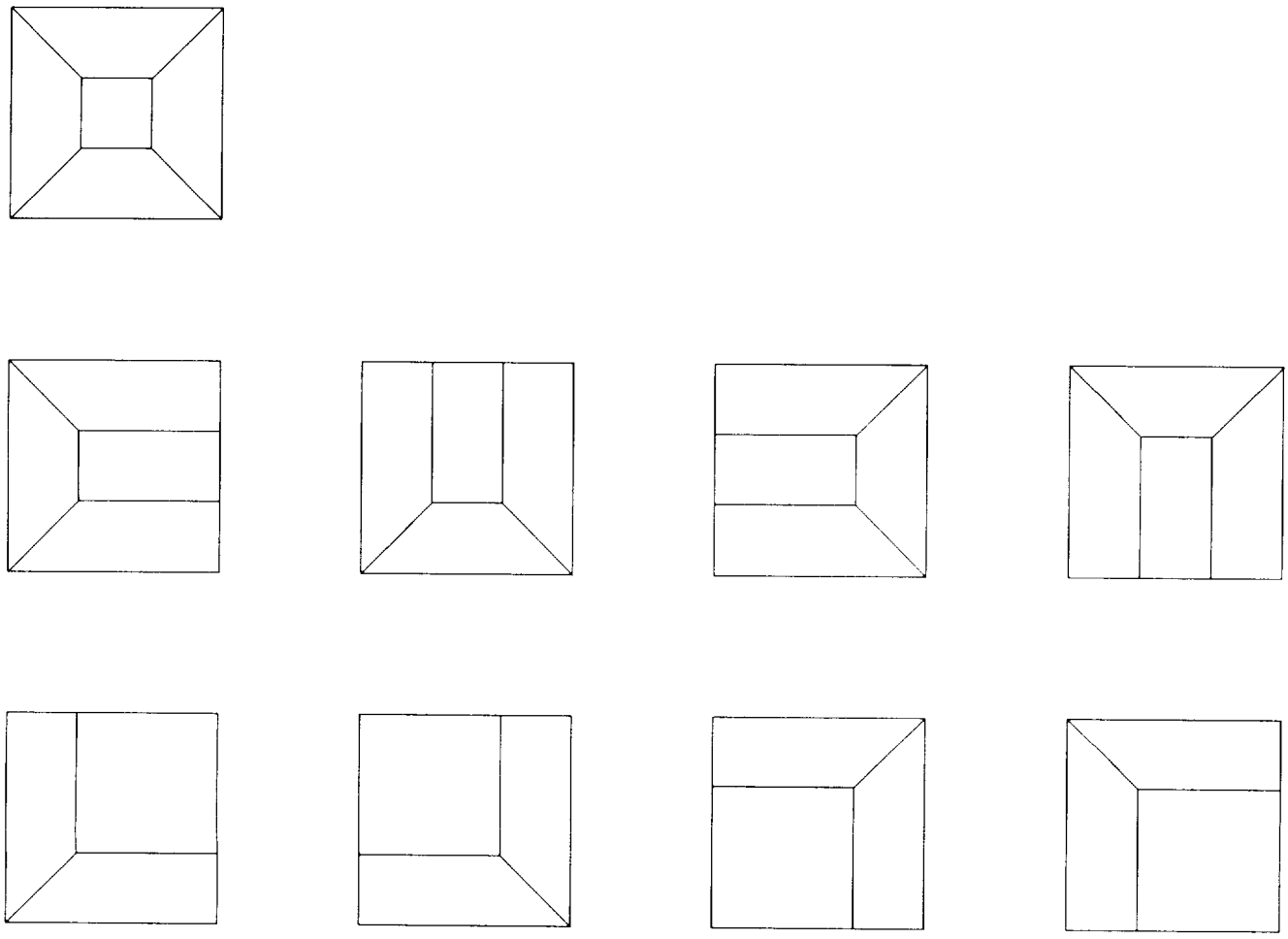
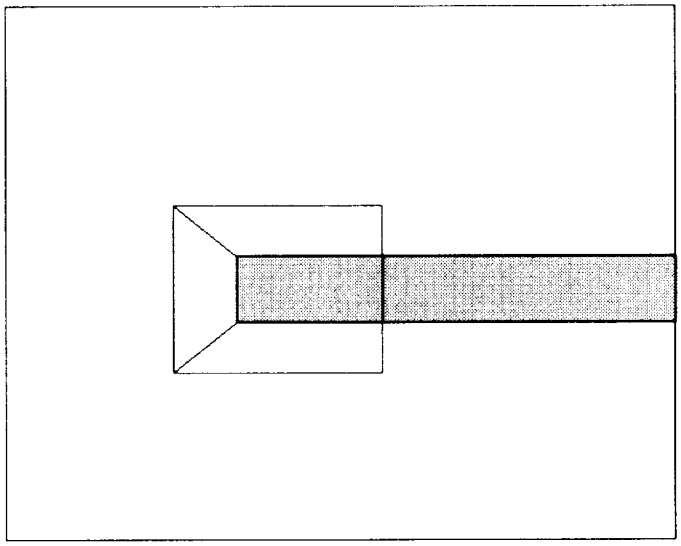
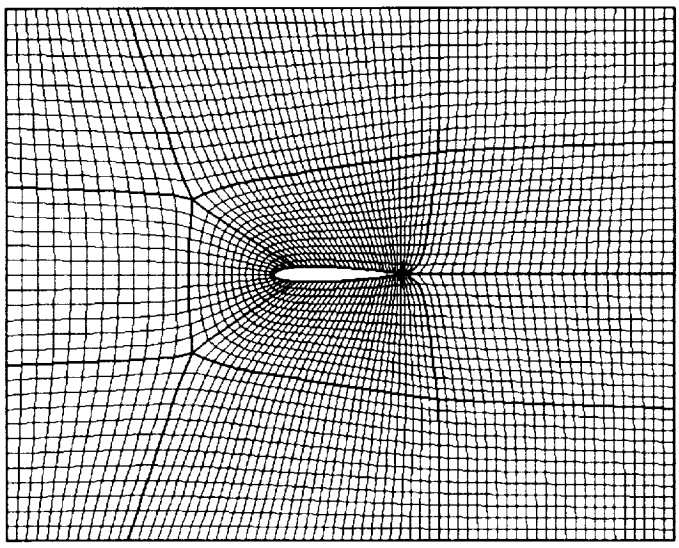


Figure 3: The nine possible wraps in two dimensions.

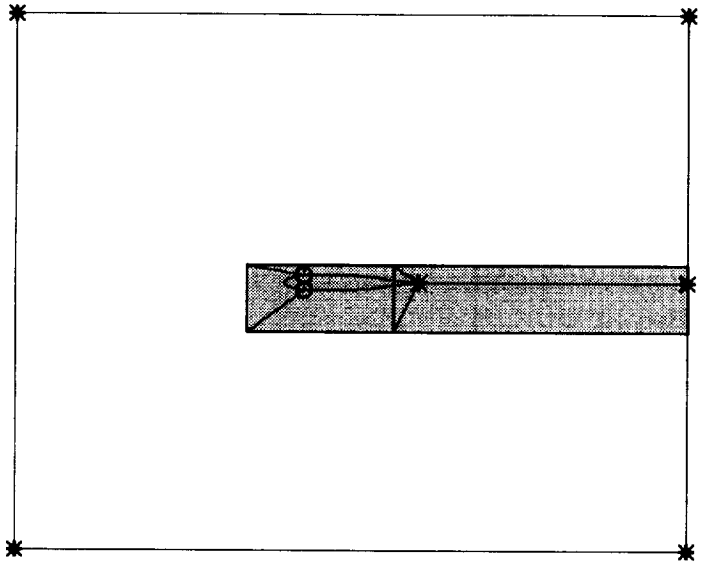


a) abstraction

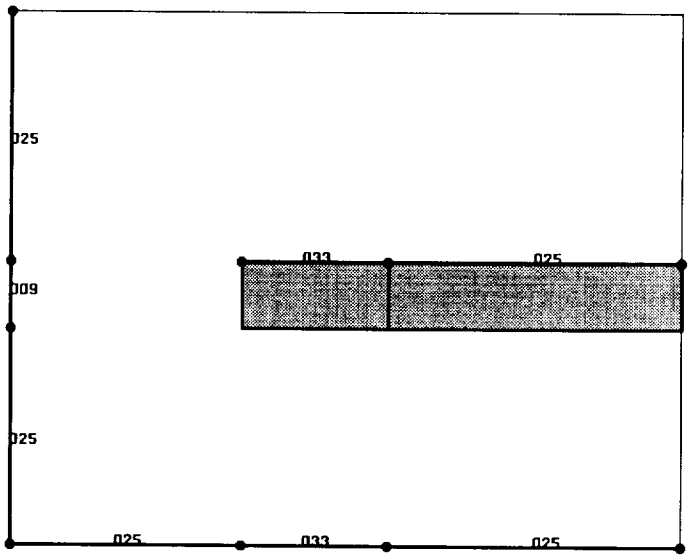


b) grid

Figure 4: CH-grid around airfoil in wind tunnel.

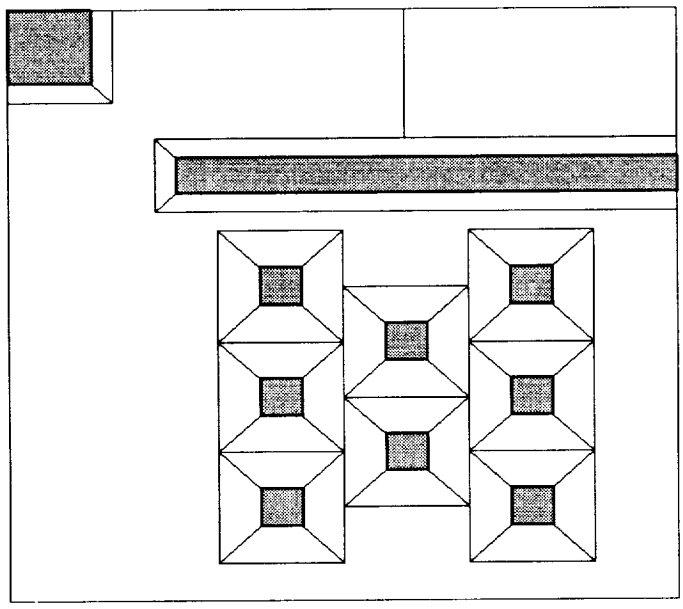


a) abstraction and configuration with attaches

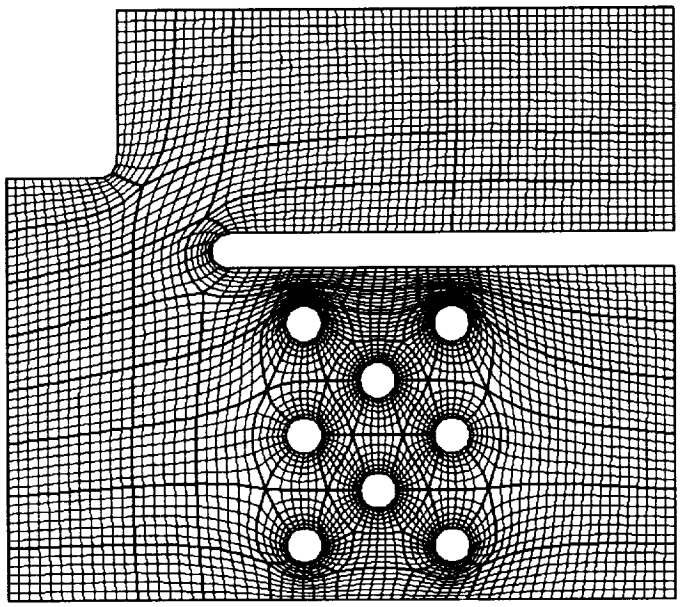



b) abstraction with rulers

Figure 5: H-grid around airfoil in wind tunnel.

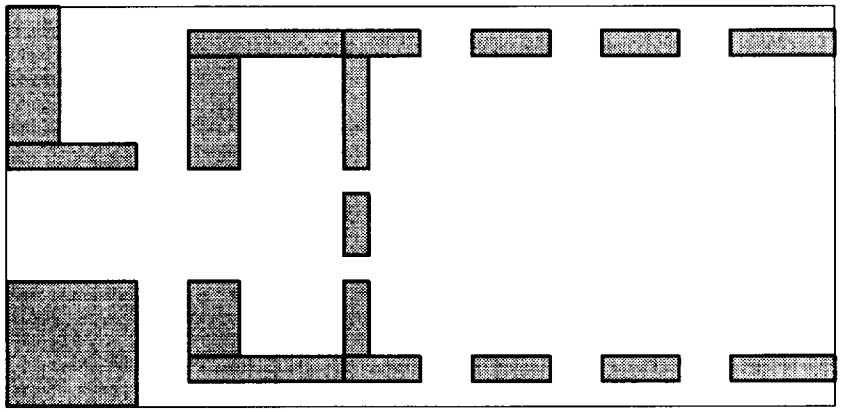


a) abstraction

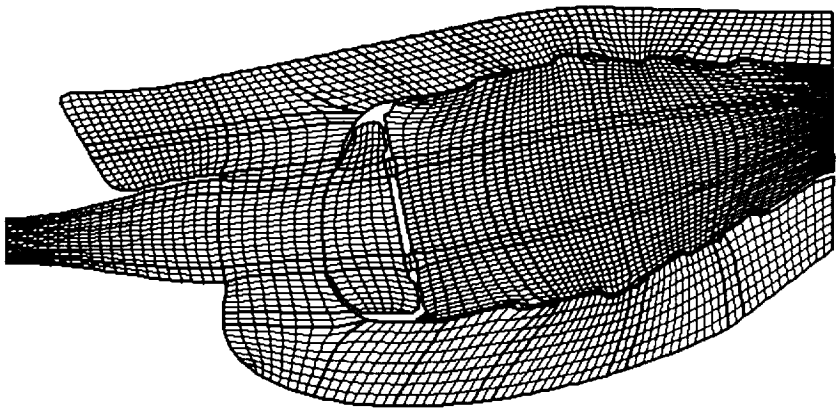


b) grid

Figure 6: Block-structured grid around passage with eight posts.

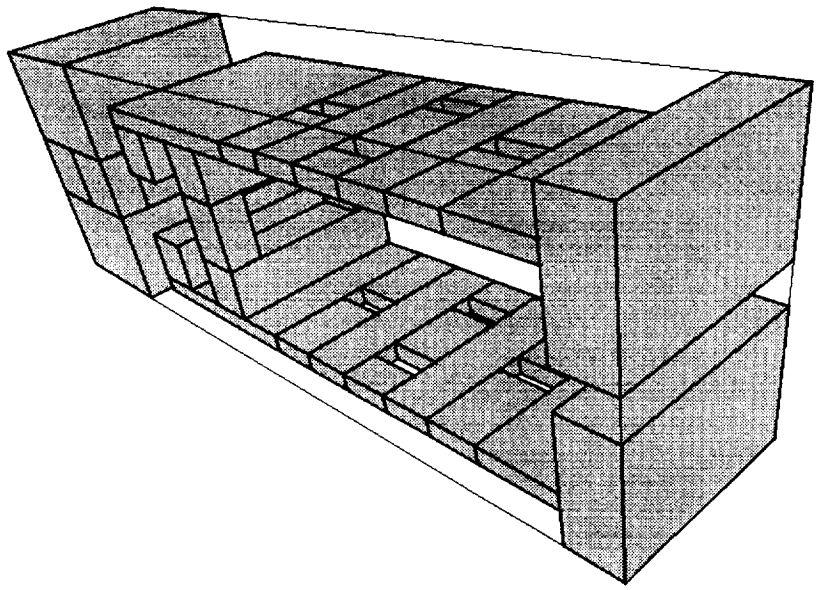


a) abstraction

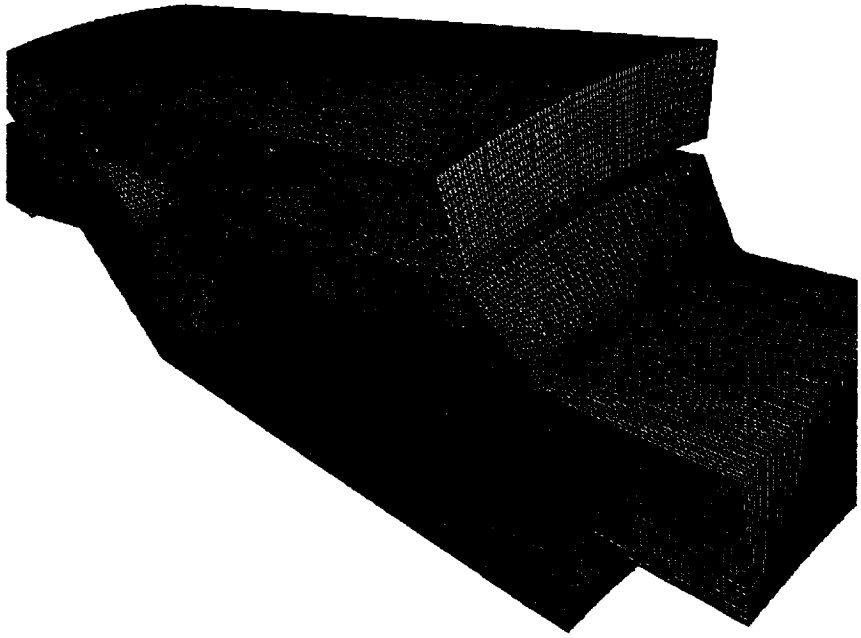


b) grid

Figure 7: Grid for combustor cross-section.

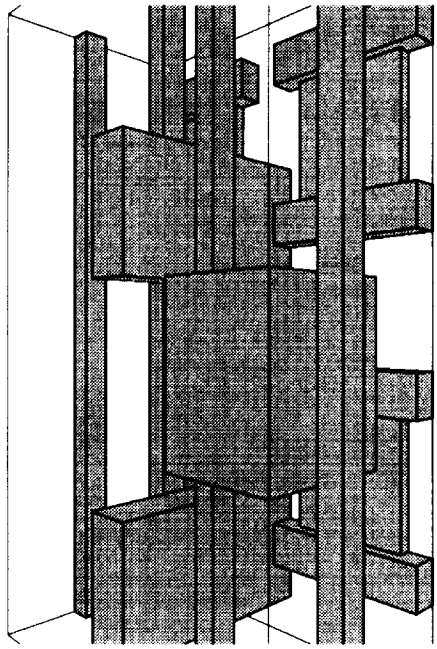


a) abstraction

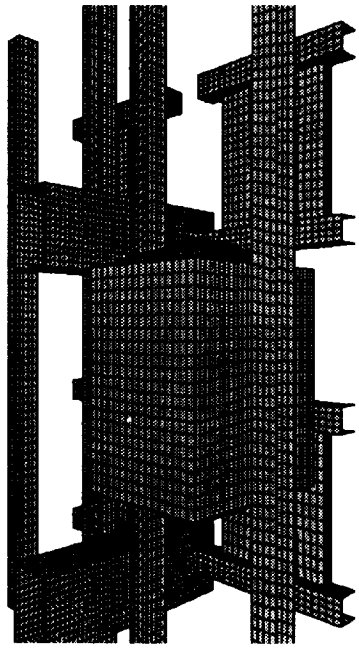


b) grid

Figure 8: Grid for three-dimensional combustor.

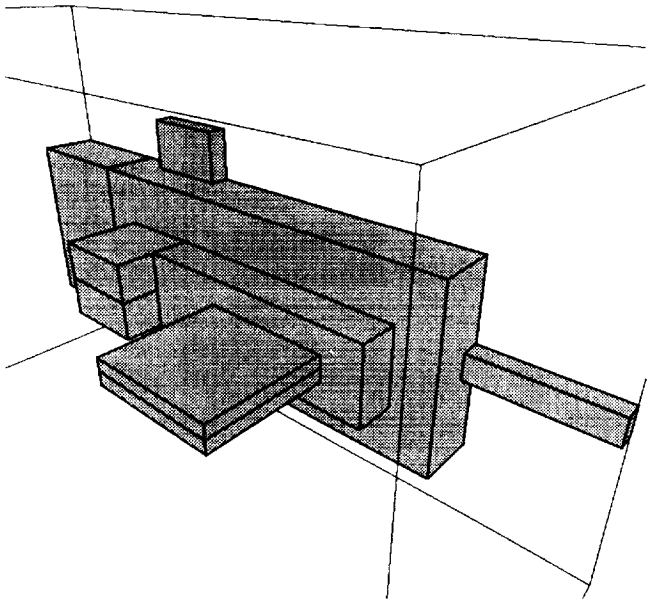


a) abstraction

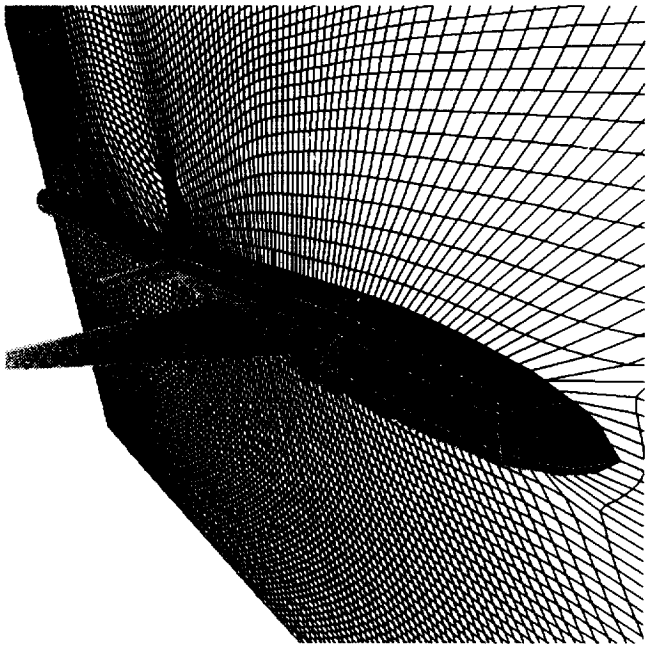


b) grid

Figure 9: Grid for elevator hoistway and streamlined car.



a) abstraction



b) grid

Figure 10: Grid for generic fighter.

N95-28735

19951223144

# MULTIBLOCK GRID GENERATION WITH AUTOMATIC ZONING

Peter R. Eiseman

Program Development Corporation
300 Hamilton Avenue, Suite 409
White Plains, NY 10601

## SUMMARY

An overview will be given for multiblock grid generation with automatic zoning. We shall explore the many advantages and benefits of this exciting technology and will also see how to apply it to a number of interesting cases. The technology is available in the form of a commercial code, GridPro®/az3000. This code takes surface geometry definitions and patterns of points as its primary input and produces high quality grids as its output. Before we embark upon our exploration, we shall first give a brief background of the environment in which this technology fits.

## MULTIBLOCK GRID GENERATION

Multiblock grids provide the best data structure for a wide variety of analysis programs. Because the local coordinate pattern of grid points is regular, the accounting overhead for point to point linkages is low and is quite simple. This means that there are more algorithms available for analysis, and moreover, that those algorithms are generally more efficient. In addition, the local coordinate structure has been recognized to produce generally more accurate results. These advantages from a single coordinate grid, carry over into the more general setting where many blocks of coordinate grid are continuously glued together to form a multiblock grid. Without a continuous glue between blocks of grid, one has the problem of transferring data between either non-aligned or overset grids. This involves many issues which relate to the management of data and the preservation of accuracy. When the glue between blocks is continuous, then the data transfer problem is vastly simplified and accuracy is greatly enhanced. The accuracy is even further enhanced as the continuity level increases (e.g. continuous derivatives). Moreover, multiblock grids are also well positioned to take advantage of parallel computing environments.

The multiblock template provides zones for the various coordinate grid blocks to reside. This is a general structure and, as such, is one which can be used for arbitrarily complex configurations. It is, in fact, a coarse unstructured dissection of space into chunks. In distinction from what is typically called an unstructured grid, the cells define macro regions in space that are called zones and which generally have curved boundaries.

## TRADITIONAL METHODS

In the traditional approach to multiblock grid generation, the creation of a grid is done in a piece by piece fashion. The process starts with the geometric region boundaries, continues by forming the zonal boundaries, proceeds by generating grids on each boundary section, and concludes with volume grid generation. The main distinctions between the

various traditional methods is the order in which the pieces can be put together, the numerical methods employed to treat each piece, the style of interactive graphics utilized to help the user deal with all of the pieces, and the ability to replay and edit parts of a session by the use of journal files.

In the course of applying a traditional multiblock generator, the user must not only insert rather large amounts of data but also must make judgments about that data. Such judgments often concern the relative spatial location of various objects. These can vary from something as basic as placing one zonal corner in space to more detailed matters such as determining both the shape and location of zonal edges. Unfortunately, the detailed judgments continue with such items as trying to appropriately adjust edge or surface grids on either side of a zone or region. Altogether, the traditional user is faced with the geometry and grid generation for corners, edges, and surfaces before he proceeds to volumes. In the usual progression of events, many judgmental errors are made and these translate into repeated operations. The result is great human inefficiency and loss of time. By the time volumes are considered, the real human work has usually been done.

## THE AUTOMATIC METHOD

With the grid generation technology to be discussed here, there is a huge paradigm shift. There is no longer the rather manual construction and assembly of multiblock grids that is fraught with the error prone tendencies of human judgment. There is no longer the massive amounts of detailed data required of the user. There is no longer the huge reliance upon high end computer graphics to manage that manual activity. Instead, the amount of input data is enormously reduced, the detailed human judgment is gone, and the management of data is efficiently controlled with an appropriate language. Moreover, the quality of grids is much, much higher. It is the same mathematical underpinnings that reduce the user effort which also account for the substantial increase in quality. Basic grid quality appears in the form of near orthogonality, smoothness, low warpage, and curvature clustering for both concave and convex boundaries.

The discussion will focus upon the program called GridPro®/az3000 which is a commercial product and which is the most automatic and powerful multiblock grid generator available. Unfortunately, there are too many grid generators that are referred to as "automatic" and, as a consequence, user's are quite naturally confused. After all, a good number of the traditional multiblock grid generators are called automatic ones. To make some sense out of this situation, the real question that should be asked is "when does it become automatic? or "from what stage is it automatic?"

As the story is examined, it will be seen that GridPro®/az3000 is automatic from an earlier stage than any other 3D multiblock generator and produces higher quality grids as well. That stage, is defined by the presence of region boundaries, the pattern of grid points, and a small handful of scalar parameters. Surface grid generation, zone construction, and the intersection between surfaces are examples of the tasks that are solved as part of the automatic solution. Of the user inputs, the primary one is the pattern of points which is referred to as the grid topology. The language that organizes the pattern of points is called the "Topology Input Language" (TIL). TIL codes are extremely flexible items and can be used to generate grids about various classes of configurations. In this context, GridPro®/az3000 compiles and runs TIL codes. While future developments are likely to include both interactive and automatic TIL code generation, our concentration shall be on the TIL language, its benefits, and the grid generation capabilities. An overview of GridPro®/az3000 is given in Eiseman, Cheng, and Häuser [1] and a detailed account is given in the user's manual [2].

## GridPro®/az3000

The practical industrial needs for engineering analysis are speed, accuracy, reliability, and realistic configurations. The latter often leads to massive complexity. Such complexity appears when configuration boundaries contain a very large number of geometric components. A large number of grid points is also desired but is balanced against the available computer resources. With massive complexity, comes the requirement to deal with an arbitrarily large number of blocks. For the traditional multiblock grid generators, this will require an enormous amount of human time, and thus, wall clock time to get the grid.

GridPro®/az3000 satisfies the practical needs of industry. Because of the high grid quality, various analysis programs have produced results on the first try, are more accurate, and are faster to convergence. Because of the automation, massive complexity is efficiently handled with a large number of blocks. The count can go to a thousand, ten thousand, or more depending upon the scope of the analysis.

## THE TOPOLOGY INPUT LANGUAGE

The scope of analysis varies from small to large as various boundary components are assembled into successively more comprehensive region configurations. In a parallel sense, the topology input language (TIL) builds successively more complex grid point patterns by assembling various topological components. With this natural building process, the basic routine in TIL codes is called a "COMPONENT". These represent both the main program and the various subprograms (e.g. like a SUBROUTINE in FORTRAN). Like most other subprograms, variables can be imported and exported from COMPONENT's. In TIL, these variables are labels for corners, vectors, and surfaces. The actual corners, vectors, and surfaces can be specified within any component. While they are common to all components, they cannot be referenced unless a correct label is available in the calling component. The output labeling convention provides this. The call of a component entitled "name" is given by the syntax of

INPUT n name(input and output labels);

where n is a positive integer tag which becomes part of the new label that is used inside the component where this input appears. As the succession of components are called into action, the labeling process continues with successive labels. Thus, a particular item may be re labeled a number of times. The re labeling will stop once the main component is reached and this will then provide a unique trace back to its point of origin. As these traces can be valuable for finding topology errors, they are included in diagnostic output that is automatically given when such errors occur.

While corners, vectors, and surfaces have been mentioned above, there was no mention of their purpose. To consider the solution of some field analysis for some region, the first item considered is its boundary. These are composed of surfaces of various sorts. These may be defined by certain analytical or piece wise analytical formulas or by some digitization. These may come from CAD systems or from other forms of construction. In the case of digitization, the surface is defined by a data file. The analytic types provide some built in types together with a location for user specification for the rest. In TIL, the surface is "defined" by only one line. This may reference a data file, provide parameters for a built-in analytical type, or may link to the location of a user specified analytic type.

With the boundaries represented by the one line surface definitions, the next items are the corners. The collection of corners defines a topology sketch. This sketch appears in physical space (not an idealized abstract space). Within the physical space sketch pad, the task is to define a coarse hexahedral wire frame. Each wire frame cell defines the topology element for a block of grid. The entire wire frame is defined by a sequence of corners and should extend over most of the space. Certain of the corners must also be assigned to surfaces in order to tell the system which parts of the topology are to be on which parts of the physical region boundary. While each corner must be put somewhere in physical space, there is no precise requirement to give a certain location. The only demand is the fuzzy topological one that the placement be in "general position". In practical terms, the corners assigned to a surface should roughly follow it to reflect major variations in surface shape.

It has been seen that the basic part of a TIL program is the pattern of points as represented by a coarse wire frame of corners with links and surface assignments. Along with the organization imposed by the use of components, there are global settings that can be made at the top of the program as well as the ability to add further TIL operatives by an include statement. The typical global parameters are the number of grid cells for each link and the dimension. The default number of cells is 8 and the default dimension is 3. The number of cells is then locally changed, as desired, within the components. Any change there automatically propagates to any effected links. The include statement permits the user to grow TIL code libraries to cover problem classes of interest. This happens naturally because COMPONENT's are reusable and can be called any number of times. The result is an efficient assembly process which can take advantage of recurrent structure. In addition, there are an assortment of other commands and options that are helpful.

## THE CORNER DEFINITION STATEMENT

The basic elements of a corner definition are its label, its position, its assignment to surfaces, and its links to other corners. Its label is just a positive integer which appears after the key letter "c" that starts the corner definition line. The only rule to be obeyed is that as corners are defined the labels must increase in size. Gaps in the labeling are permitted and are sometimes desirable. The gaps can be useful when one wishes to subsequently edit the component by inserting some new corners between two previous ones. For cosmetic reasons, they may even help to add human clarity to a TIL component.

After the corner label is given, the next item in the corner definition is the position in physical space. In a direct sense, this is given by a sequence of three numbers to reflect the ordering for x, y, and then z. The delimiter between each item on the line of definition is just a space: there are no comma's! While such a direct definition is often convenient, it is not as flexible as the one which comes from vector operations. The "x y z" sequence is replaced by "@ vector operations" where the symbol "@" is the flag to tell the system to expect the operations with vectors. In practical terms, this allows one to create corners that are automatically positioned relative to other corners, be they within the current component or imported. Moreover, the positioning can be even more arbitrarily accomplished by the use of vectors that are either defined in the current component or imported from the outside. While all corners are vectors, the vectors are more arbitrary since they are not required to convey the specific details associated with corners.

After the corner position is given, the next item in the corner definition is the surface assignment. If there is to be a surface assignment, then the flag "-s" is given. After a space delimiter, the desired surface labels are given with spaces between them as delimiters. The order in the list of surface labels is unimportant. Each label points to a

surface definition statement which in turn provides a desired surface association. A rule that must be obeyed here is that the surface definition must already exist. There is simply no meaning if one should designate a surface that is not there! The available surfaces for assignments are those which are either defined in the current component or which are defined by an import into the current component. The latter is slightly more abstract and also more general. To provide for an import, the surface is defined in the component argument list. However, it will not really exist unless an actual surface label is inserted in that spot when the component is used. If the spot is given as a black in the component application ( i.e. in the INPUT statement), then the assignment to will also be a blank. This, however, offers the opportunity to design components which can be applied in many more situations.

After the surface assignments are given, the final item is the specification of linkages to other corners. If there are to be a links to other corners, then the flag "-L" is given and is followed by a list of the corner labels with spaces for delimiters. The order in the list is again unimportant. The only requirement is that the corners must already exist by either a local definition within the same component or by a defined input to the same component. As in the case with surfaces, there is the opportunity to design more generality into the components by using blanks when there is no corner link at the INPUT stage.

The end of line for any TIL statement is given by a ";" and the corner definition is such a statement. To illustrate the corner definition syntax, an example is given. Consider the statement

c 5 1.2 3 4.5 -s 1 3 -L 3 4 2;

This defines corner 5 to be located at the position (1.2, 3, 4.5), to be assigned to surfaces 1 and 3, and to be linked to corners 3, 4, and 2. Notice that there is no detail about how the links to other corners are made. For simplicity, they can be considered as straight lines. This lack of detail is what distills the wire frame down to the level of just a sequence of corners. Also, one may notice that there is no detail given for the surface assignments. The surface labels point to the surface definitions which are single lines. A block face will be placed upon a particular surface if the corresponding four topological face corners are assigned to that surface. Moreover, a block edge will be placed upon the intersection of two surfaces if the two topological corners for the edge link are assigned to the two surfaces. This is how the user gets the intersections between surfaces.

## SURFACE DEFINITION STATEMENTS

The basic parts of a surface definition statement are the label, the type, the parameters or file name, the orientation, the cluster intensity, and transformations. Spaces are the delimitors between the various parts and associated parameters. The surface definition line starts with the key letter "s", has the delimitor of one or more spaces, and is followed by the label. The surface labels are positive numbers given in an increasing order. Gaps in the numbering are allowed and can be desireable for certain occasions. After the surface number, the definition type is given and is followed by its parameters or the associated file name, as is required of the specifed type. Beyond this are the other items. The orientation is given by the direction of the unit normal vector to the surface. The rule is that the normal vector must point into the region to be gridded. As there are two sides to any bounding surface and as each such surface is given a natural normal direction, there is only the need to reverse the orientation should the natural normal vector violate the rule. Orientation reversal is given by the flag "-o"

With the properly oriented bounding surfaces, the next items are to cluster to that surface or to transform it. The flag for clustering is currently given by either "-c" or "+c" to denote minus and plus cluster groups, respectively. The actual clustering for either group is given as a real number which follows the flag after a space or so. The transformation flag is "-t" or "-R" for a directly stated translation or linear transformation, respectively. Both can be used in the same statement. The translation is given by a succession of three real numbers while the linear transformation is given by a succession of nine real numbers. However, when vectors are used both are combined into one item which is expressed as "-t @( )". Within ( ) a single vector is inserted if it is only a translation and four vectors are inserted if it is a general transformation. In the general case, the first vector is the translation while the remaining three are the respective rows of the 3x3 linear transformation. Because vectors can be defined elsewhere, this is a variable assignment. Aside from these options attached to the surface definition, there are a few remaining technical options which are not frequently used.

To illustrate the surface definition syntax, a few examples are given. First consider the statement

s 2 -plane(0 0 1 -2) ;

This defines surface 2 as the plane z=2 with orientation in the positive z direction. The type is "-plane". The first three numbers represent the normal vector (0, 0, 1) while the last gives the constant value of 2. The ";" gives the end of line. Because of the orientation, the grid is assummed to be above the plane.. The plane is defined by the function

$$f(x,y,z) = ax + by + cz + d$$

as the level surface f(x,y,z)=0. Level surfaces of f are those surfaces which are determined by constant values of f. Since f=c is the same as (f-c)=0, it is always possible to replace f with f-c ; and thus, there is no loss of generality in considering only the constant values of 0. With the same constants (i.e. 0), the expression alone then carries the total definition of the surface.

As a second example, consider the statement

s 5 -ellip(0.5 2 0.25 3.2) -o -t 0 0 4 ;

This defines surface 5 to be of (super-) ellipsoidal type. The first 3 parameters are the recipricals of the semi-axis lengths while the fourth is the exponent which determines the degree of squareness. Thus, the semi-axial lengths are 2 along the x-axis, 0.5 along the y-axis, and 4 along the z-axis. The squareness of 3.2 is greater than that of 2 which would define a pure ellipsoid. The natural orientation which points to the outside of the super-ellipsoid is reversed with "-o" in order to consider a grid on the inside. With the basic "-ellip" type being defined about the origin, it is lifted up by 4 units in the z direction with the translation vector (1, 0, 4). For surface 5, the parameters are a=0.5, b=2, c=0.25, and n=3.2 in the function

$$f(x,y,z) = |ax|^n + |by|^n + |cz|^n - 1$$

which defines the level surface. As n varies from 2 to infinity, the surface varies from a pure ellipsoid to a brick. It is also easy to see that a natural normal direction is in the outward convex direction since this is also the natural direction of the gradiant of f. It is also worth noting that the multiplicative nature of a, b, and c is quite convienient since cyclinders can also be readily specified by setting any of these parameters to zero.

## SURFACE GEOMETRY INPUT

The surface geometry can enter the process in a wide variety of forms and must be distinguished from any particular form. This is especially true if the geometry is given by a point data set.. Often a misinterpretation can occur because the set appears in the format of a surface grid. The necessary distinction is that this input grid is only used as a data set to define geometry: there is no requirement to place points at any particular position for the act of grid generation. The only requirement is to represent the geometry accurately enough to permit a decent grid generation. This means that there must be an accurate enough resolution of curvature.

In the current implementation, GridPro®/az3000 can take both implicit and explicit surface data types. The implicit surfaces are the level surfaces. They are implicit since the evaluation of points on such surfaces comes from the solution of an algebraic equation f(x,y,z)=0. By contrast, points on explicitly defined surfaces are evaluated directly from their parameters. That is, for a given (u,v), the surface point is just (x(u,v), y(u,v), z(u,v) ). In many instances, there are a number of sections for an explicit surface. The evaluation of a point then involves a search to find the desired section after which the direct evaluation transpires.

The built-in implicit surfaces are

(1) planes
(2) super-ellipsoids, and
(3) cylinders

In the TIL syntax, the surface definition types are denoted by "-plane ', "-ellip", and "cyclind" respectively. Examples of the first two were given in the previous section. The third type is a cyclinder in the form of a periodic surface.

As already noted, the explicit surfaces have a parametric nature since surface points are determined by local coordinates once the local surface segment is found. The current ones are

(1) A single coordinate grid
This has bilinear coordinate parameters for each element and is brought into action with the type flag "-linear".

(2) Multiple coordinate grids
This has bilinear coordinate parameters for each element and is brought into action with the type flag "-compos".

(3) Unstructured quadrilateral elements
This has both a plain format and a MSC/NASTRAN format in the forms CQUAD and CQUADR. It is brought into action with the type flag "-quad".

(4) Unstructured triangular elements
This has both a plain format and a MSC/NASTRAN format in the forms of GRID, CTRIA3, and CTRIAR. It is brought into action with the type flag "-tria".

(5) Surface of revolution around an arbitrary center curve

This is given by a data file which lists a sequence of ordered quadruplets (x,y,z,r) where the each (x,y,z) is a centerline point and r is the radius of the circular disk normal to the centerline point. The first line in the file is the total number of quadruplets which is just a positive integer. This type is brought into action with the flag "-tube".

Unlike the implicit types, the explicit ones require data files for the surfaces. After the type indicator, each data file is given between quotes. Without extra options, these appear in the surface definition statements of TIL in the format

s n -type "name.dat" ;

for surface number n. Here -type is any of -linear, -compos, -quad, -tria, or -tube as described in the above list. After the type designation flag, the user chooses "name" for the corresponding data file "name.dat".

The built-in implicit types are very convenient to apply since surface definitions are given by a small number of defining parameters. While additions to the list of built-in implicit types will add further convenience, there will always be cases that are not covered. Thus, provision has been given to user defined implicit types. The window of opportunity for non-built-in type appears in the form of a C-code that is supplied to the user. For a fixed surface, the user simply writes out the equation for the level surface desired. This writing is done in only one specific location. In the case where periodicity conditions are necessary, more locations must be used to prescribe those conditions. All of this data is inserted into a file called "name.h" where the user chooses "name" and inserts his data in the right spots. After the type flag (-implic) in the surface definition line, the user gives "name.h" and continues with other items to complete the line.

In a slightly more abstract sense, a floating surface can also be defined. It is abstract since there is no defined position or condition for this surface. This type of surface is used to define clustering to any block face regardless of where it is. This clustering can be applied to either side of the surface or to both sides. To bring this into action, the flag "-float" is used in the surface definition line; next, there is a cluster group flag (either "-c" or "+c"); and then there is the desired spacing for the first grid point on either side. Once the surface definition line exists, the clustering takes effect when the corners of a chosen block face are assigned to this surface. If clustering is desired for only one side of a floating surface, then the sides are selected by inserting a number 0 or 1 just after the type flag "-float".

Altogether, we have 3 modes of surface prescription. The first is a surface that has a fixed position in space. The second is the periodic surface which seeks its own position in space to optimize grid quality while accommodating periodic boundary conditions. The third is the floating surface which has no conditions other than the clustering specifications attached to its definition.

Unlike other schemes, there is no dependence upon the details of surface geometry definition. The geometry is simply viewed for its trace of points in (x,y,z) space. There is no concern for any specific surface parameterization and how it might abut another set of surface parameters. However, there is a need for the geometry to be well defined. A well defined geometry is one which gives an accurate specification of the solid objects. It must be free of constructive irregularities such as unexpected holes which would make hollow objects out of ones that should be solid. In additon, if two independently defined surfaces are to intersect, then they must actually do so in the region of interest. Otherwise, one is left with holes that ill define a solid object. A further desire is for the geometry definition

to be compatible with the chosen pattern of grid points. This situation can occur if adjacent grid sheets are chosen to wrap over a region of sharp surface curvature. In such cases, it is best to have a smooth surface even if that smoothness is only seen on a micoscopic level. An alternative is, of course, to select a different topology. Altogether, because of the freedom from detailed geometry description, virtually any form of surface definition can be addressed and built-in. This provides a healthy growth path.

## THE SCHEDULE FOR RUNNING THE GRID GENERATION

Once the TIL code is finished and put into a file, the next task is to run it. This is done by creating a schedule file for the run. The schedule can vary from a straight run to one which is dynamically changed. The option to change the course of a run allows the user to optimize the run.. For example, the run could be started with a fairly modest number of grid points which are then increased in stages as the grid is relaxed towards its equilibrium that progressively satisfies the underlying variational scheme. As one could imagine, there are a number of parameters in the schedule that provide the user helpful options. One of these provides the bounds for curvature clustering. This is in effect a geometry adaptive process. Upon activation, the volumetric grid points distribute themselves about the boundaries in proportion to the boundary curvature.

## FUTURE DIRECTIONS

With the component structure of TIL, there is the opportunity to make TIL components in an object oriented sense. This can quite naturally feed into an interactive environment where the topology set up can be done. Given the non-unique nature of grid topology selection, some choices are clearly better than others. These can vary with the physical problem or region geometry. This will lead to the application of expert systems to help in the guidance. In addition, there is also the consideration for a turn key creation of the coarse wire frame for topology definition. This would then automatically generate the TIL code and thus the grid. The automatic generation of the coarse wire frame can certainly be done with today's methods. Here, one simply does a coarse unstructured tetrahedral grid generation (via advancing front, octtree, etc.) and then breaks up each tetrahedron into 4 hexahedrons. While this approach will certainly define a topology through a coarse wire frame, it will also yield too many severe singularities. Thus, there is a need to do the unstructured generation with more quality. This means the automatic generation of hexahedral grids with appropriate quality and desirable pattern for the targeted applications. Examples of pertainent activity are paving (2D) and plastering (3D). Altogether, these various directions for topology generation are in the realm of research and development.

## EXAMPLES

In this section, we will look at a selection of examples. These will cover a cross-section of industries and show the type of grids that are created. This will explicitly show the quality level. Beyond these pictures, it is natural to ask for the amount of computer resources that are needed as well as the length of the TIL code. These items will be given. The next question may very well be to estimate the length of human time. This is more difficult since the user must give a region geometry, create a topology and cast it into TIL. In fact, geometry specification and topology creation are a part of any multiblock grid generator. In GridPro®/az3000 the ease with which various topologies can be considered also means that the user has a more plentiful supply of choices. While each choice can be addressed with a traditional multiblock grid generator, it is so painful to do so that it is, for

all practical purposes, not a real option. A typical case of this sort is the application of compact enrichment. Here, coordinate surfaces are made to stay within the local region requiring enrichment. In effect, certain surfaces are steered back to the same subregion. In the case of geometry specification, GridPro®/az3000 can accept exceedingly general surfaces as input. This includes surfaces digitized in an unstructured format (e.g. NASTRAN) as well as implicit surfaces. The use of implicit type surfaces is extremely powerful and general. An examples of it is given in the companion paper by Cheng and Eiseman [3].

The examples given here start with a blade row for turbomachinery. Then a grid is generated inside a single turbomachinery blade. This illustrates the effect of clustering to concave and convex curvature in the simple 2D setting. Next, the same sort of curvature clustering is witnessed in the 3D case of a automobile fuel tank. The fuel tank geometry was given by an unstructured triangular mesh while the fuel surface level was given by an implicit form for a plane. After this, compact enrichment is viewed with a case for an airfoil over ground. Again the simplicity of 2D makes it easy to see the concept of compact enrichment where coordinate curves are steered back to surfaces. This case also shows a branch cut off of the trailing edge. In continuation, compact enrichment is examined in 3D with a grid for the ONERA M6 wing. In distiction from the prior case, this shows the enrichment in the setting where coordinate surfaces wrap around the entire wing. The enriched locations are around the entire perimeter of the wing planform. Shifting our focus back to automotive engineering, we next examine the case of an air induction system and then a two port cylinder. This is then followed by a manifold configuration which is a case with four tubes emerging from a large chamber. Each of the tubes emerges from the chamber with an abrupt intersection. Once again, we return to compact enrichment and look at only one tube abruptly intersecting the chamber. The enrichment appears as a collar about the curve of intersection and can be easily witnessed in the figure. While the last two cases examine abrupt intersections, there are parallel cases with filletted intersections. Fillet creation and its use is examined in Cheng and Eiseman [3].

## CONCLUSION

The power of multiblock grid generation with automatic zoning has be explored along with the associated program: GridPro®/az3000. The diverse nature of the options and applications have been witnessed as has the flexibility of the program. Some future directions have also be charted for the creation of still other programs the generate input for GridPro®/az3000. Altogether, the future for this technology is very bright.

## REFERENCES


1. Eiseman, P. R., Cheng, Z., and Häuser, Applications of Multiblock Grid Generation With Automatic Zoning. Numerical Grid Generation in Computational Fluid Dynamics and Related Fields, Ed. by N. P. Weatherill, P. R. Eiseman, J. Häuser, and J. F. Thompson, Proceedings of the 4th International Conference held at Swansea, Wales, 6-8 April, 1994.

2. Program Development Corporation, GridPro®/az3000 User's Guide and Reference Manual, Program Development Corporation, White Plains, NY, September 1994.

3. Cheng, Z and Eiseman, P. R., Examples of Grid Generation with Implicitly Specified Surfaces Using GridPro®/az3000: Filletted Multi-Tube Configurations, this NASA Conference Proceedings, May 1995.

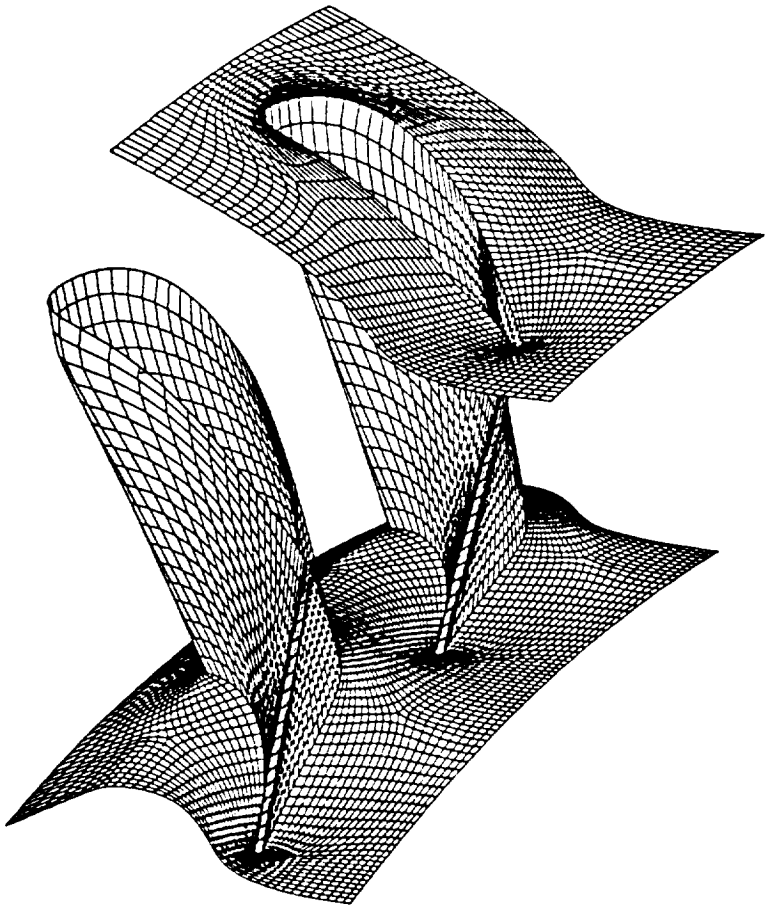


**Figure 1**: A row of turbomachinery blades. The choice of toplogy provides line of sight from upstream and downstream directions. That is, coordinate curves go directly from these directions onto the blade surface. The grid pattern appears as a polar like wrap for the blade which is smoothly integrated into a streamwise type grid. Periodic boundary conditions are employed around the axis. Those boundaries come from the user specification for the number of blades. Grid orthogonality and smoothness are maintained regardless of solidity: the blade spacing and angle of attack can be made arbitrarly severe without deterioration in grid quality. The TIL code is one page and the computation time is less than one hour on a medium grade workstation.

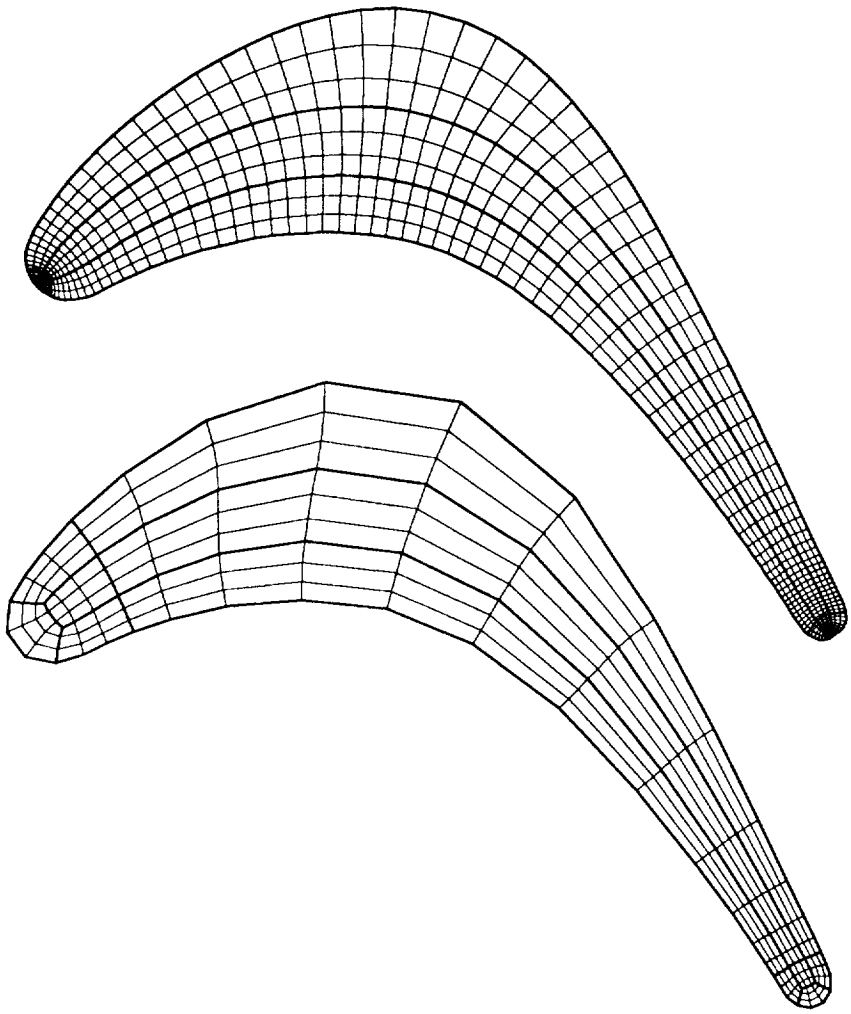


Figure 2: Inside a 2D blade contour. The effect of curvature clustering can be readily witnessed. The most intense clustering is at the blade trailing edge, the next most intense clustering is at the leading edge, and is followed by lesser clustering on the top and bottom of the blade. In contrast, the grid is thinnest at the location where the blade is nearly straight (i.e. flat). While most of the curvature is concave the bottom of the blade is the only convex part. To see how **GridPro/az3000** deals with a small number of grid points, the second grid shows the same attributes as the fine one. The TIL code is one half a page and the computation time is less than one hour on a medium grade workstation.

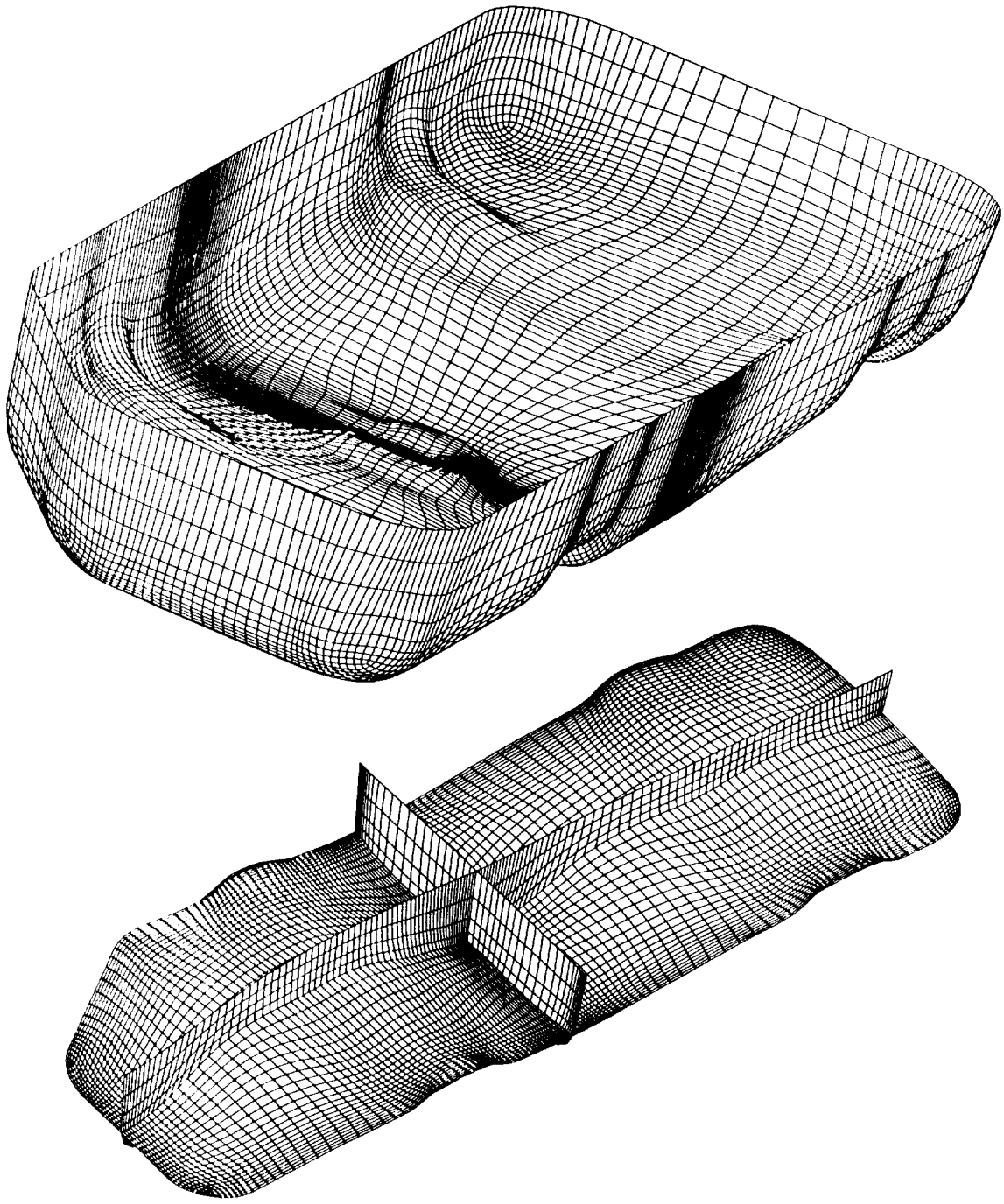


**Figure 3**: A half full fuel tank for an automobile. This grid is for a crash analysis application.(with LSDYNA3D). The fuel surface is given by a plane as a built-in implicit type. The fuel tank geometry is given by an unstructured triangular mesh (e.g MSC/NASTRAN type CTRIA3) and has both concave and convex curvatures. Those curvatures appear at the tank corners and at the locations of the drain and of the straps which hold it in place under the vehicle. Clustering to both concave and convex curvature is easily seen as is the smoothness and near orthogonality. Because of the application in the area of structural analysis, the multiblock output of **GridPro/az3000** was expressed in NASRTRAN format. The TIL code is less than a page and the run time is about one half an hour on a medium grade workstation.

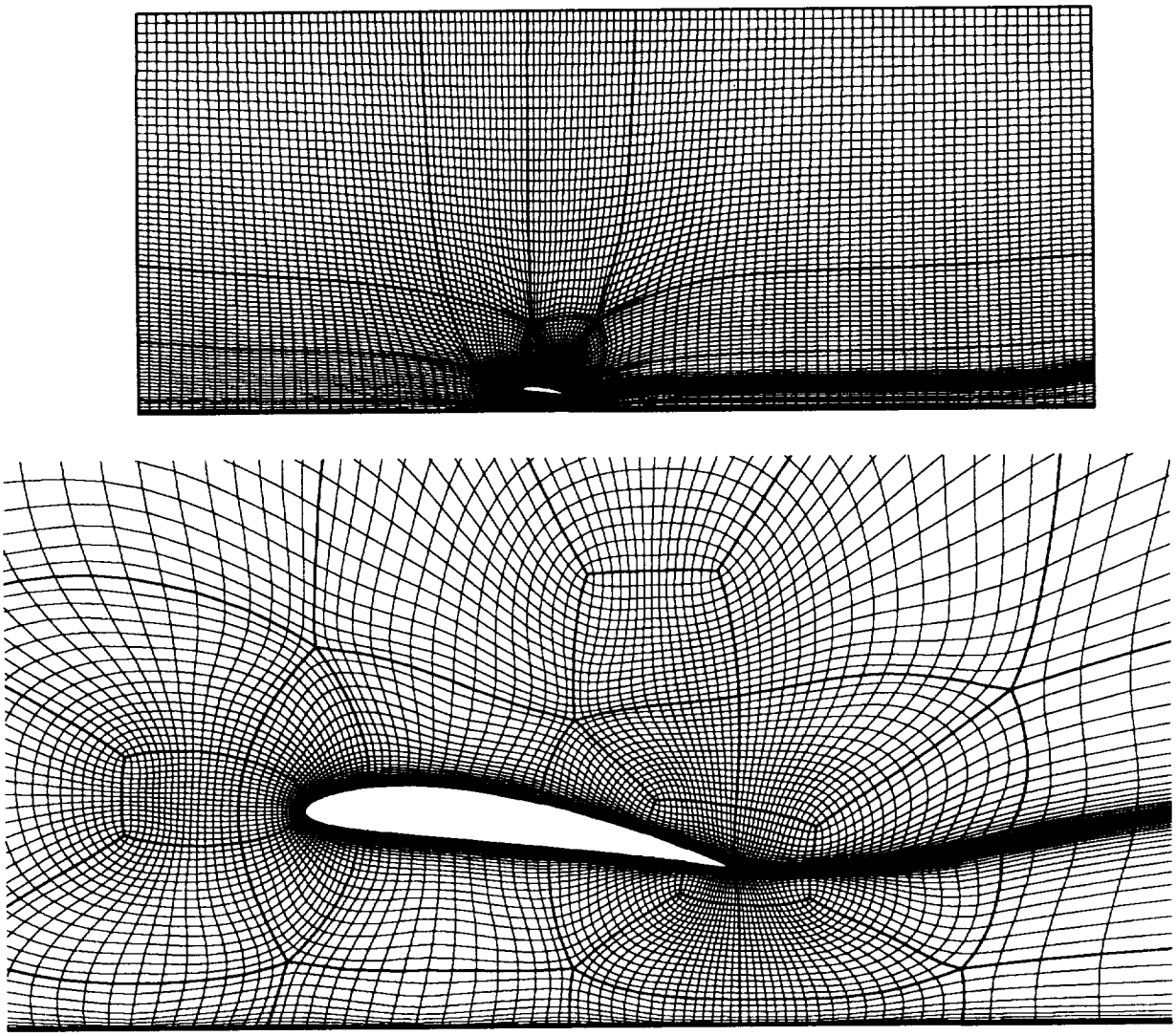


Figure 4: An airfoil close to the ground. The grid is for a study of ground effects on the perfomance of an airfoil. The airfoil contour is given by a digitized data set. This is an example of the use of compact enrichment. In the global picture, it is clear that there are no undesireable clusters of points on the boundaries either above or upstream of the airfoil. While a cluster appears for the full length of wake branch cut, it too could have been limited to some distance before the downstream boundary if desired. However, this option was not chosen. In the vicinity of both the leading and tailing edge, the coordinate curve density is at its highest. One can readily see that the chosen pattern of points (grid topology) sends curves looping from the bottom of the airfoil to the top about both leading and trailing edges. It also can be seen that similar loops appear from the top of the airfoil and end upon the ground surface. By contrast, the bottom of the airfoil was left to be that of a simple block. We often call such loop structures "clamps" or as in the case of the trailing edge structure "a collar." The TIL code for this case was a little over one page. The run time is less than one hour on a medium grade workstation.

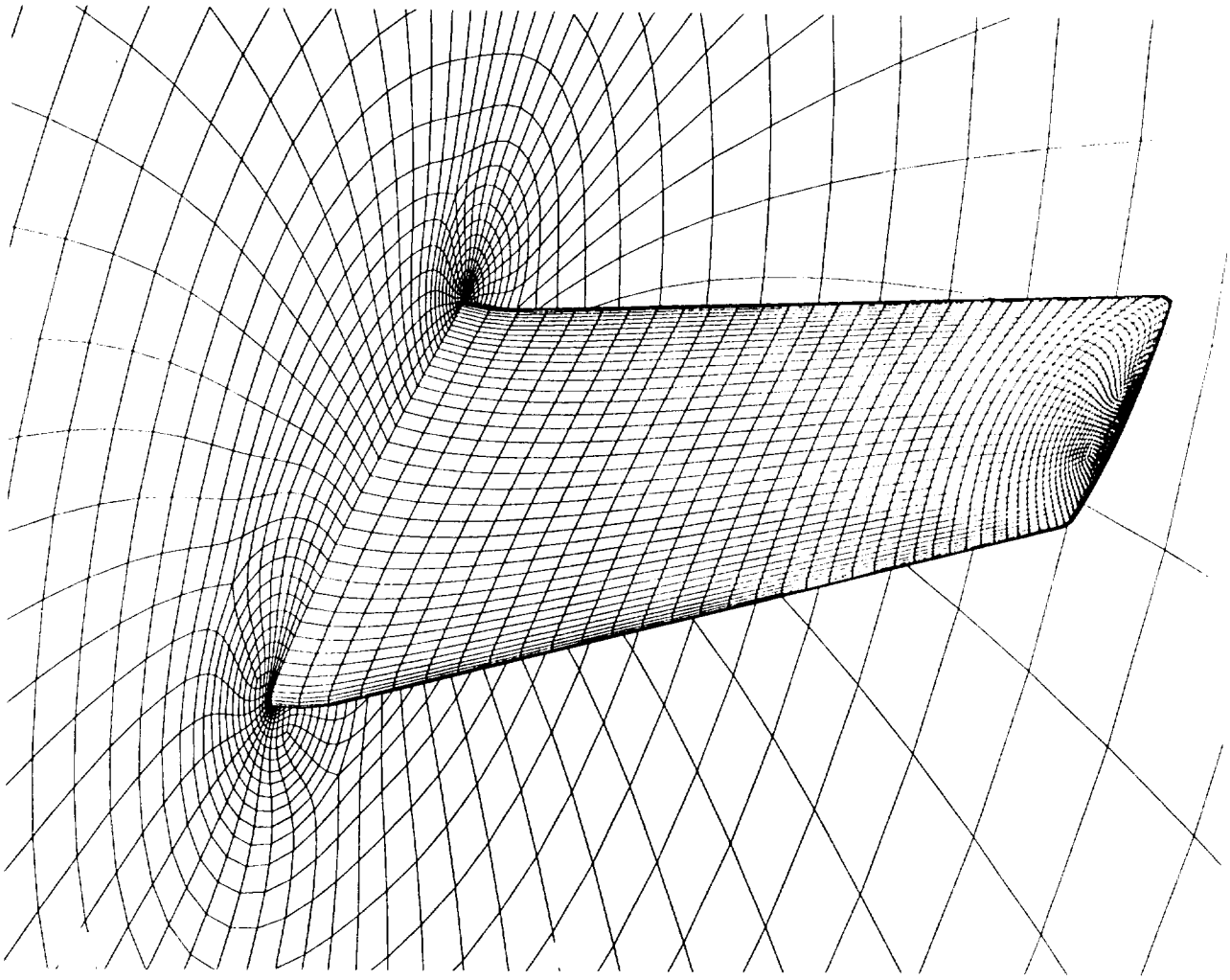


**Figure 5**: The ONERA M6 wing. This is a standard test case for the study of a basic wing. The basic grid topology is chosen to wrap around the whole wing. The pattern can easily be seen in the symmetry plane. Within the wrap around structure, compact enrichment has been applied to the entire perimeter of the wing planform. In the symmetry plane, it is easy to detect the compact enrichment at the leading and trailing edges. That enrichment appears in the form of curves that loop over a regular block and connect the wing surface with itself. We call this structure a "clamp." It is this compact structure which is carried from the leading edge up to and around the wing tip and then back to stop at the symmetry plane trailing edge. Altogher, the looping coordinate curves look like the shells of a distorted half torus or, more figuratively, like the surface of a half an automobile tire. The geometry of the M6 wing was given in a digitized form and the TIL code was about two pages. The grid generation was less than one half an hour.

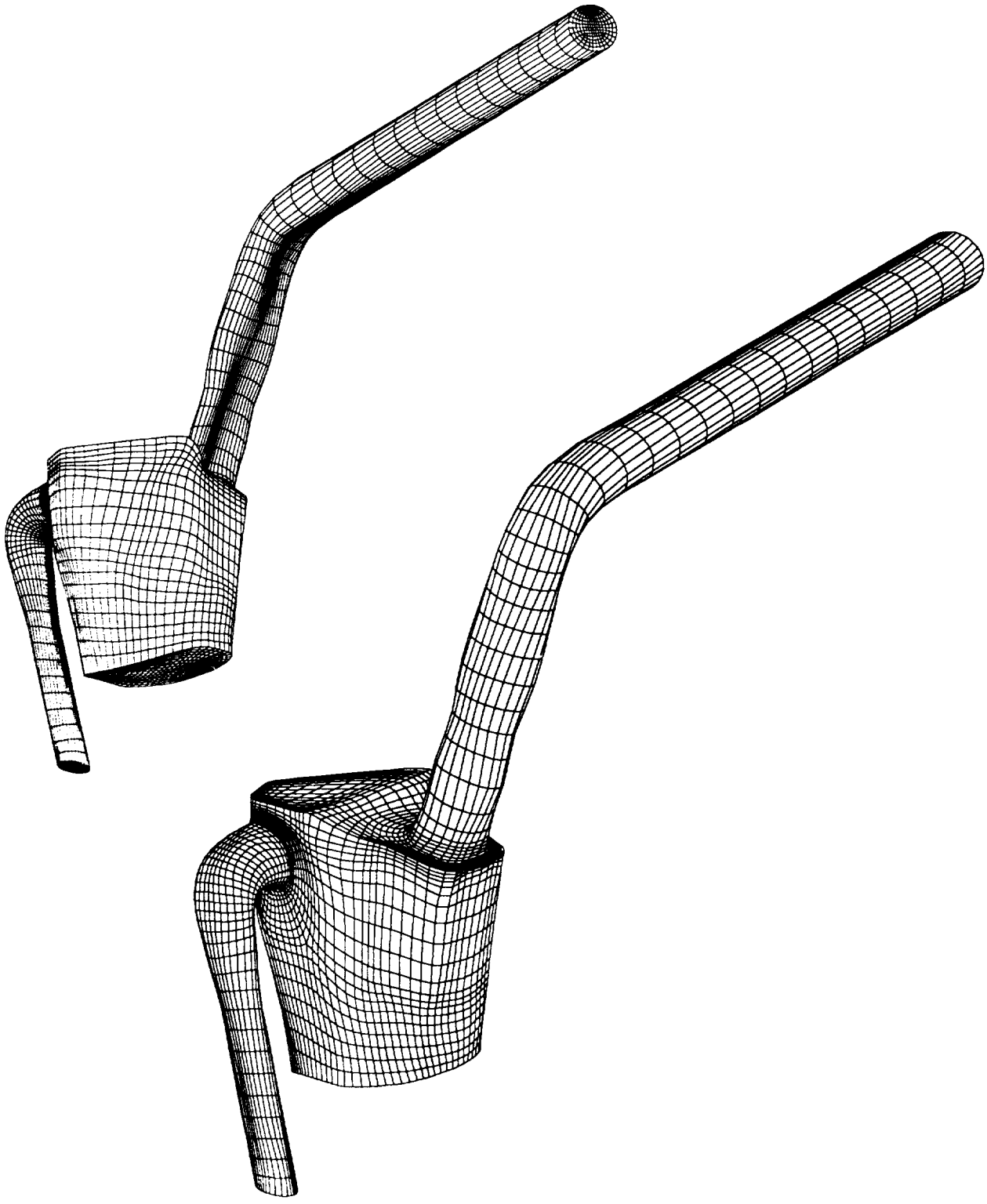


**Figure 6**: An air induction system for an automobile. This is an example of one sort of plumbing that is common in the automobile industry. The surface was given in by an unstructured quadrilateral mesh in MSC/NASTRAN format. The TIL code was 4 pages and the run time about 15 minutes.

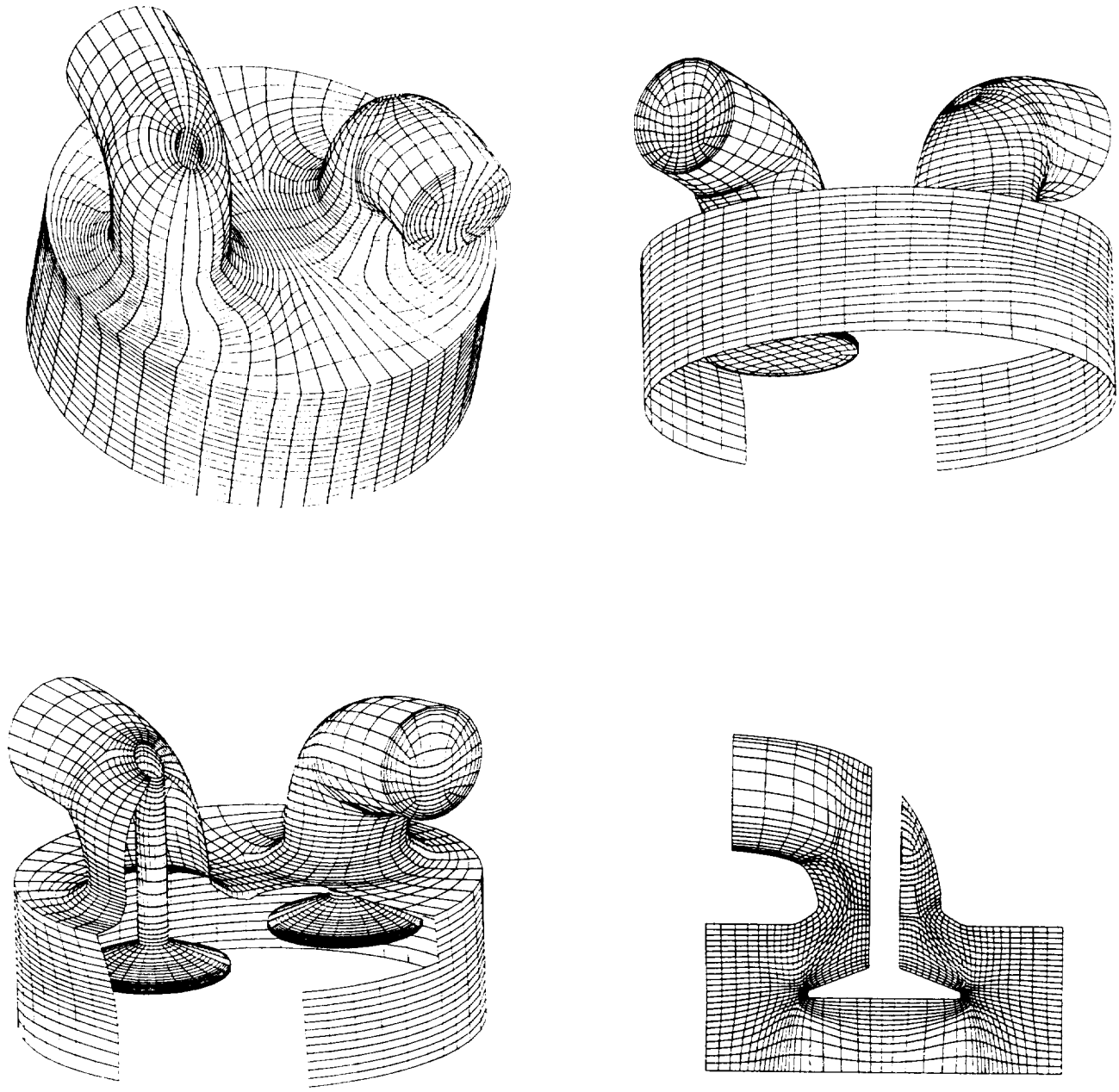


**Figure 7**: A two port cylinder for an automotive engine. This is a generic case for the class of two port automobile cylinders. In a prior application, there was a need to generate a grid in a pipe bend that had a rod inside it that started in a parallel manner at the entrance but continued straight up and intersected the pipe as it went through the bend. The topology for that prior case was a single page of TIL code and was in a separate file. Noting that the prior case could be also used for the port in the pictured application, the first line of the TIL code was an INCLUDE statement for this file. The remaining TIL code was just a few pages. The generation of the grid was about 10 to 15 min of computer time.

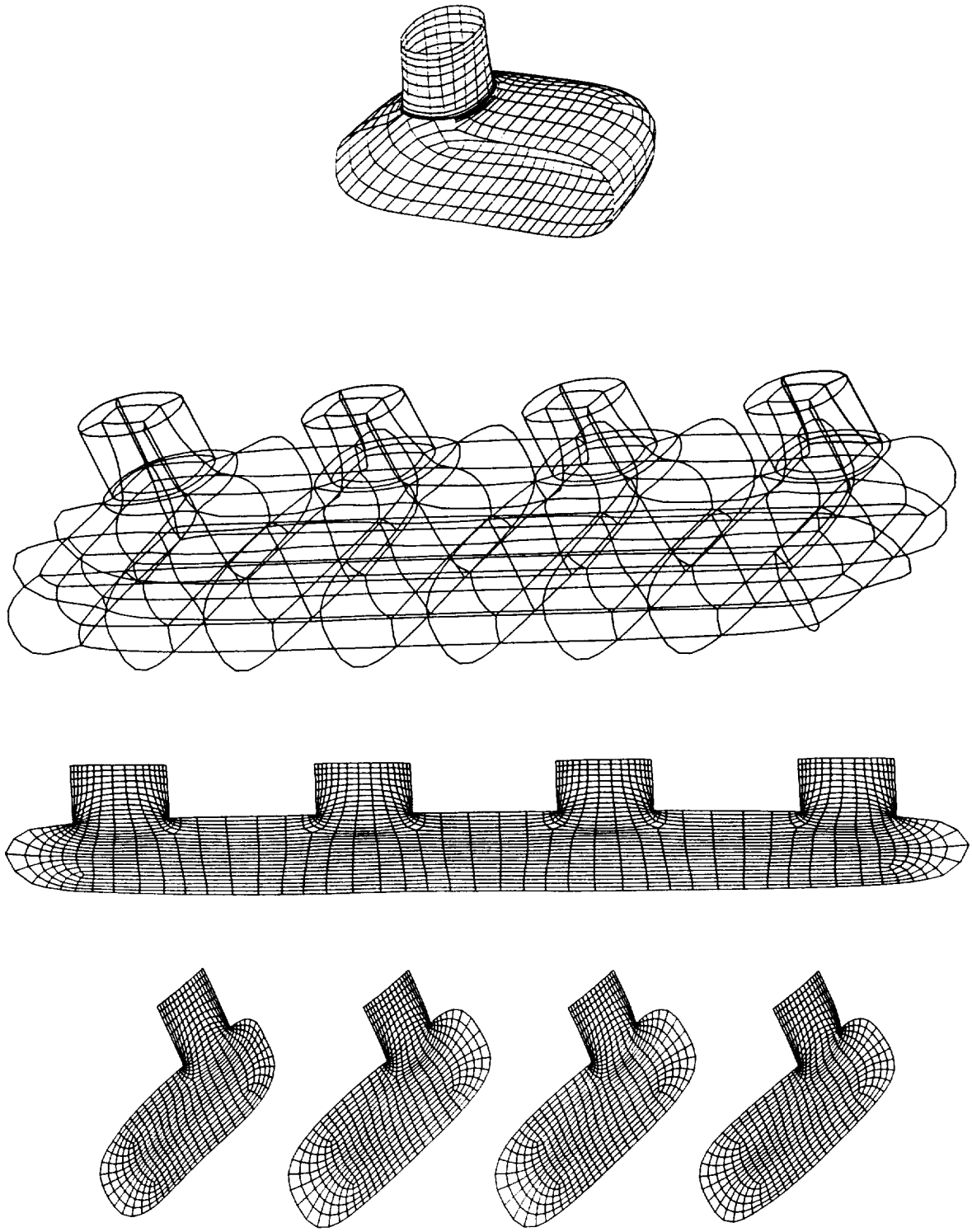


**Figure 8**: A generic manifold for an automobile engine. The geometry consists of four tubes which abruptly intersect an elogated chamber from above. With the freedom to choose a generic geometry, the chamber and the tubes were rapidly assembled by using implict functions of the type "`-ellip`." The TIL code was 4 pages and the grid generation was about 15 min. on a modest workstation.

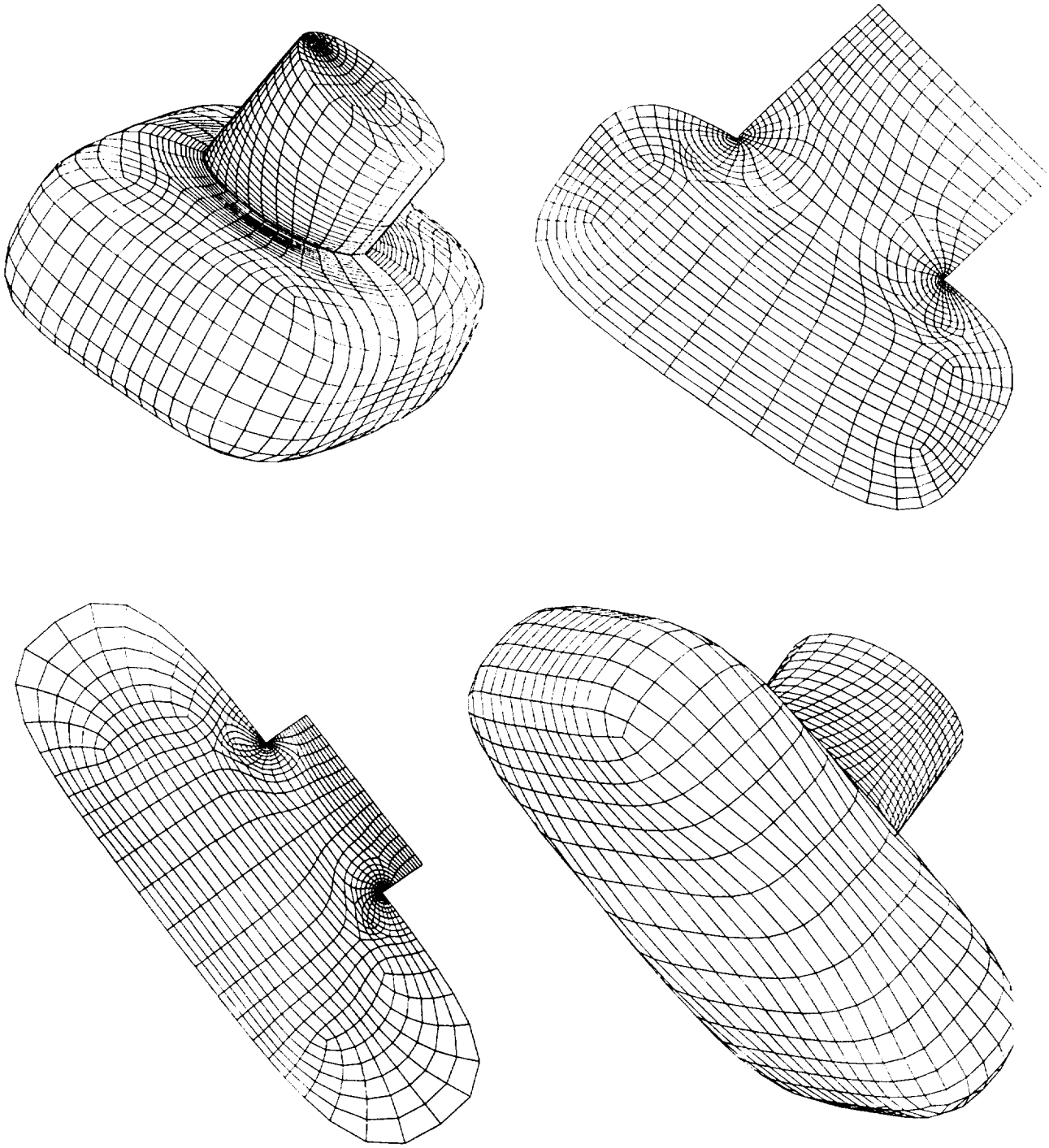


**Figure 9**: An abrupt intersection between a tube and a chamber. In the context of an abrupt tube intersection, the element of compact enrichment is shown. This is seen in the form of the "collar" structure which wraps around the entire curve of intesection between distinct surfaces. In addition, the associated TIL code was developed in an object oriented style. In so doing, the length of TIL code was longer than the normal few pages. However, the "objects" are very flexible in their reuse and adjustment. The main component in this TIL code was then reduced to about on fifth of a page. Major configuration changes could then be done with just a few in line alterations. In this case, such change are the addition of exta tubes into the chamber. It is this style which is the seed for a more complete object oriented library in TIL. With this type of library in place, the role of interactive graphical assembly can be empowered.

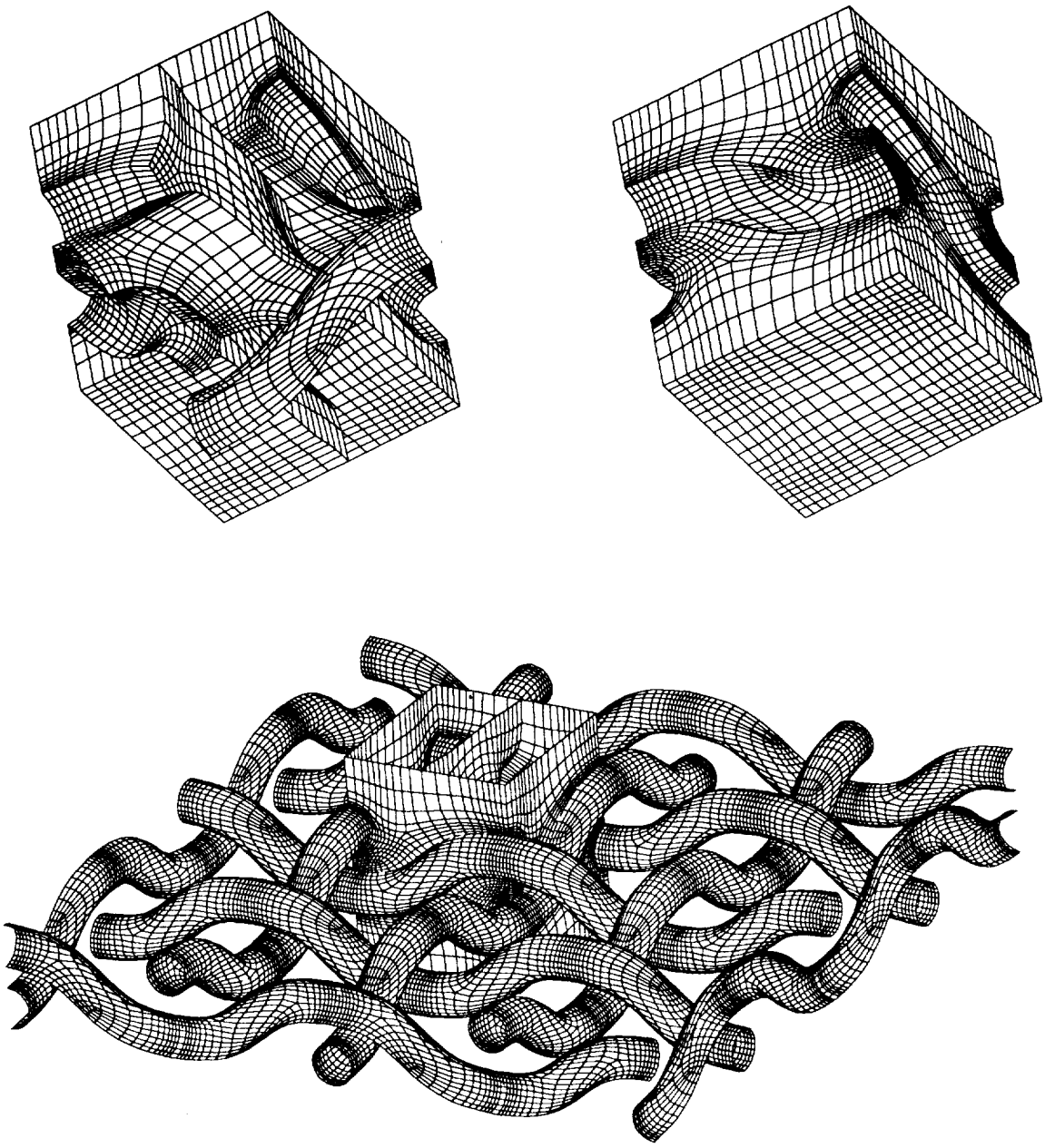


Figure 10: The field about cloth fibers. The grid is generated in the region about interwoven fibers that are bounded by planes above and below the cloth. This configuration has applications in filtration, in molding processes, and in the apparel industry. The geometry of the fibers is given by the type "-tube" with the actual tube geometry given by a data file of centerline coordinates along with cross-sectional radius. While this case was executed for fibers of a constant radius, a simple change of data file is all that is needed to do a variable radius. The grid is readily seen to be smooth and nearly orthogonal. The TIL code is two pages long and the run time on a slow workstation is about 15 min.

N95-28736

19951223 15

# BLOCK-STRUCTURED GRIDS FOR COMPLEX AERODYNAMIC CONFIGURATIONS: CURRENT STATUS

Veer N. Vatsa, NASA Langley Research Center, Hampton, VA
Mark D. Sanetrik, AS&M, Hampton, VA
Edward B. Parlette, ViGYAN Inc., Hampton, VA

## SUMMARY

The status of CFD methods based on the use of block-structured grids for analyzing viscous flows over complex configurations is examined. The objective of the present study is to make a realistic assessment of the usability of such grids for routine computations typically encountered in the aerospace industry. It is recognized at the very outset that the total turnaround time, from the moment the configuration is identified until the computational results have been obtained and postprocessed, is more important than just the computational time. Pertinent examples will be cited to demonstrate the feasibility of solving flow over practical configurations of current interest on block-structured grids.

## INTRODUCTION

The field of computational fluid dynamics (CFD) is rapidly approaching the stage where simple configurations, such as wing-body configurations, are routinely analyzed and designed at cruise conditions with CFD tools. However, the situation is not as promising when one considers a more complete configuration, especially at take-off and landing, where high-lift devices must be deployed. The difficulties arise mainly because the task of generating the grids for modeling such complex geometries is tedious, and because the computing time associated with obtaining flow solutions on associated grids containing millions of grid points is excessive. To a large degree, the difficulty associated with generating structured grids for complex geometries has made the unstructured-grid approach more attractive. During last few years, unstructured-grid-based methods have been shown to be very flexible and powerful tools for analyzing inviscid flows by solving the Euler equations. For viscous flows, where the solution of Navier-Stokes equations is needed to capture the flow physics, further research is required to make the unstructured codes more efficient.

Structured-grid-based methods offer a viable approach for solving viscous flows over complex configurations, but have not realized their full potential due to difficulties associated with grid generation. The principle advantage of such methods lies in the efficiency of the associated algorithms in obtaining a solution once a grid has been established. The arithmetic operation count to attain a solution with such algorithms has seen steady decline since early days of CFD research. The advancements in raw speed of the computers have further decreased the computational resources required to obtain converged solution to the governing equations. The difficulties encountered in grid generation can be overcome to a large degree by employing the "divide and conquer" philosophy in constructing block-structured grids for analyzing complex configurations, where each block (or set of blocks) focuses on a given component or region of the domain. Recent work on overlapped, block-structured grids, where structured grids are generated around individual components, has demonstrated the applicability of such an approach for analyzing some of the most complex aerodynamic configurations (ref. 1). The principle drawback of this approach is that conservation is not enforced in the overlapped regions in most codes. From the standpoint of conservation and accuracy, it is desirable to enforce point-to-point match ($C^0$ continuity) across block interfaces. However, such a requirement imposes severe constraints on the grid-generation process. A reasonable compromise can be achieved by accommodating different grid densities at the block interfaces without the overlapping of grid lines. Such an approach is commonly referred to as a patched-grid approach, and can alleviate many of the difficulties associated with the grid-generation process. In the present paper, we will summarize our experience with the use of block-structured grids for complex configurations.

## OVERVIEW OF CURRENT CAPABILITY

This section is divided into two parts that deal with grid-generation and flow-solver related issues, respectively. The following discussions are kept brief intentionally, because more detailed information on these topics can easily be found in the open literature.

### Grid generation

There is no dearth of block-structured grid-generation schemes for complex configurations, as is evident by the large number of participants in recent conferences on grid-generation (refs. 2–4). Almost all organizations involved in CFD simulations have some type of in-house grid-generation capability. For example, AGPS at Boeing, MACGS at McDonnell Douglas, UNISG at Rockwell/FIAT, and MBGRID at Canadair, are some of the better known grid-generation systems currently in use at various industries. In addition, commercially available software packages, such as GRIDGEN, ICEM/CFD, NGP, and GRIDPRO/AZ3000 are constantly upgraded to meet customer needs. It is difficult to list all such activities, instead we briefly discuss the general features of some of these tools from a CFD user's perspective.

As researchers associated with the CFD Laboratory at NASA Langley Research Center, we are required to analyze flow over a variety of configurations that are of interest to our internal and external customers. Most of the time, these customers provide us with only a coarse point definition of the individual components that comprise the configuration being analyzed. On few occasions, a standard CAD definition ( e.g. in IGES format) may be available. Our task as CFD engineers is to provide accurate flow solutions over such configurations in the shortest possible time.

A suitable grid must be generated before the flow solutions can be computed for a given configuration. The grid-generation task generally turns out to be the most time-consuming part of the process, particularly in terms of man-hours. Given enough lead time, most of the commercial grid-generation systems cited earlier in this paper can be used to create good quality grids, especially by their originators. However, these software packages are quite complex to use and invariably require an expert user (who probably does grid generation for a living) to create good quality grids in a timely manner. The fact that these tools are heavily interactive, with a large number of steps and paths that are traveled via complicated menu-driven branches, keeps the novices at bay. This difficulty can be overcome by creating a center of expertise in an organization, but then one has to face the consequences of compartmentalizing one facet of the solution process from the rest. It is well known that the grid density is determined by the particular algorithm and application, which in turn requires coordination between grid-generation and flow-solution and analysis phases. Such arrangements may not be practical except in large, production-oriented organizations. The point to be emphasized here is that until the grid-generation process is simplified enough to be usable by an average engineer involved in flow simulation activities, CFD as a discipline will not achieve its full potential.

The grid-generation process starts with surface modeling, after the initial geometric definitions become available. These geometric definitions can take many forms, ranging from parametric surface definitions, such as NURBS, to a collection of points measured directly from a physical model. If only a coarse point definition is available, which is generally the case, smooth and accurate surface interpolations are required to create an enriched surface with high fidelity. In general, CAD-based systems, such as ICEM/CFD, have a definite advantage at the surface modeling stage. Usually the surface definition of only the individual components of the configuration (e.g. wing, fuselage, tail ) is provided, and it is left up to the CFD engineer to determine the intersections of the various components. Accurate description of the intersections is essential for maintaining the fidelity of the underlying surfaces. Once again, CAD-based systems are generally more flexible and accurate for computing surface intersections of geometrically complex components. Such flexibility comes at the added cost of acquiring the expertise (through user training) required to navigate through a CAD system, which is generally too complex for average CFD engineer to maintain proficiency. Since surface modeling is only a small part of the total

grid-generation process, a more cost effective way to accomplish such a task is through maintaining a center of expertise, such as GEOLAB at NASA Langley Research Center.

After the surface modeling is completed, the detailed grid-generation process begins by selecting grid topologies for the surface and field grids. For multiblock grids, this requires splitting the physical domain into different blocks (zones), a process which is commonly known as domain decomposition. Most grid-generation packages rely very heavily on interactive, user-supplied intuition to arrive at the blocking strategy (domain decomposition), both at the surface and field grid levels. The interactive approach provides utmost flexibility to the user because it allows full control of blocking strategies. However, such an approach is very tedious, especially for complex three-dimensional configurations, since accurate visualization of the block boundaries in physical space is cumbersome. This is mainly due to the large number of blocks required for gridding complex configurations and due to limitations of available visualization software. To exacerbate matters, especially for $C^0$ continuous grids, a change in block structure intended for a given zone can cascade into a series of changes affecting several other zones, increasing the effort required to complete the grid-generation task.

Based on our experience, a batch-oriented approach is needed to make the block-structured grid-generation process more attractive to engineering users. The interactive means, especially graphical user interfaces (GUI's) can and should be used to help the user set up the input to the grid-generation code and to help make decisions related to topology and domain decomposition. But, whenever an interactive module is invoked, an easily understandable and editable script file should be created so that it can be added to the batch input file later on. Of course, the user must be able to specify grid density and grid spacings in various zones and subzones. Recently, a batch-oriented grid-generation package with semi-automatic blocking capability was developed by Eisemann and co-workers (ref. 5). The domain decomposition in their software package (known as GRIDPRO/AZ3000) is accomplished through specification of the block structure in a parametric topological space. The zonal boundaries in the physical space move freely and evolve simultaneously with the solution to the partial-differential equations that govern the grid coordinates. The resulting grids are relatively smooth and nearly orthogonal, except near the singular points formed by corners of the grid blocks. Although it is a promising method, in our opinion, several shortcomings of this approach (addressed later in the paper) need to be rectified before this software package can be used routinely by CFD engineers working on aerodynamic flow problems.

Before closing the discussion on grid generation, it should be mentioned that the specification of boundary conditions, especially the block-interface conditions, should be integrated with the grid-generation process. Without such a coupling, too much user time is spent on preparing the input to flow solvers, and the chance of errors increase rapidly with the increase in the number of zones (blocks) and the complexity of the grid topology. This type of capability would also reduce the need for human intervention in linking the grid generation to flow solvers, which would form the core of a complete CFD analysis software system.

## Flow Solvers

During last few years, there has been a steady growth in the number of computer codes capable of solving Reynolds-averaged Navier-Stokes (RANS) equations for steady flows over complex configurations on block-structured grids. Several multiblock codes developed at NASA centers, most notably OVERFLOW, NPARC, CFL3D, INS3D, PAB3D, and TLNS3D have been distributed to various industries, universities, and other federal laboratories throughout the United States. In addition, the large aerospace companies, such as McDonnell Douglas, Boeing, Northrop, Grumman, Rockwell, General Electric, and Pratt & Whitney, have their own customized codes. Researchers at the European Space Agency (ESA), DLR (Germany), and RAE (England) have also developed similar codes for their own internal use. The capabilities of the various codes differ somewhat, depending on the original application for which each code was developed. However, most of the mature codes are capable of handling the complex aerodynamic configurations of practical interest if suitable grids are available.

Before proceeding further with the discussion on flow solvers, it is appropriate to classify different types of block-structured grids, based on the connectivity present at the block (zonal) interfaces. In overlapped (overset)

grids, the grid lines from adjoining blocks form an overlapped region instead of a clearly identifiable zonal interface. In patched grids, the adjoining blocks form a common interface, but the point distribution on the block boundaries that form the interface is different. In $C^0$ continuous grids, the adjoining blocks that form the interface have a point-to-point match at the zonal interface. The overlapped grids, patched grids, and the $C^0$ continuous grids form a hierarchy of structured grids requiring increased effort in grid generation; the $C^0$ continuous grids require the most effort to generate. However, with regard to flow solvers, $C^0$ continuous grids are the simplest and most convenient to deal with.

The overlapping grids are by far the easiest type of grids to generate; therefore such grids have been used for computations on the most complex geometries to date, such as the flow over the complete space shuttle configuration (ref. 1). However, conservation is not enforced in the overlapped regions of the grid; therefore this methodology may not be appropriate for problems in which accurate capturing of shocks and shear discontinuities is crucial. Furthermore, the actual process of using overlapped grids requires several nontrivial steps beyond the generation of grids around individual components. For example, one must generate collar grids at wing/fuselage juncture regions for adequate resolution of the boundary layers. In addition, a variety of preprocessors are needed to determine the connectivity and interpolation coefficient matrix for the overlapped grids, which requires significant human intervention before CFD analysis can be performed.

The next step in the block-structured grid hierarchy is the patched-grid approach, where the zonal boundaries do not overlap but have different grid distribution in the adjacent zones. Considerable research has been done on patched-grid algorithms to achieve conservation at zonal interfaces. Although a conservative algorithm can be devised more easily for patched interfaces than for overlapped grids, this task is still not trivial, especially for three-dimensional curved interfaces (refs. 6–9). Due to the difficulties encountered in maintaining conservation at zonal interfaces, most general-purpose 3–D flow solvers employ non-conservative formulation at patched interfaces.

For reasons discussed previously, use of the $C^0$ continuous meshes is preferable whenever possible to guarantee conservation across zonal interfaces. Admittedly, this shifts the burden of CFD simulations to grid generation. We realize that the generation of $C^0$ continuous grids for complex configurations is a difficult task; however, the advantages of using such grids in terms of global conservation, free-stream preservation, and simplicity of zonal interface boundary conditions can be significant. For example, devising a procedure to check the self-consistency of zonal interfaces in such grids is fairly straightforward. Availability of such diagnostic tools that pinpoint user-input and grid-related errors is extremely helpful to CFD engineers. However, depending on the level of complexity of the geometric configuration, the $C^0$ constraint on grids may need to be relaxed, especially to make more efficient use of grid points and to improve the overall grid quality (smoothness and orthogonality). By placing the patched-grid interfaces away from strong discontinuities, the errors caused by the loss of conservation at such boundaries can be minimized.

## SELECTED APPLICATIONS

A central-difference, finite-volume, multiblock Navier-Stokes code TLNS3D-MB, developed at NASA Langley Research Center, has been employed to obtain flow solutions on several configurations of practical interest. The thin-layer form of the Navier-Stokes equations is used for modeling the mean flow. Unless stated otherwise, the flow is assumed to be fully turbulent, and the effect of turbulence is modeled through the eddy-viscosity hypothesis. A five-stage Runge-Kutta time-stepping scheme with three evaluations of the artificial dissipation terms computed at the odd-numbered stages is used to advance the flow solution in pseudo time. Implicit residual smoothing is used to increase the stability of the time-stepping scheme. Further enhancement in convergence of the time-stepping scheme is achieved via a multigrid acceleration technique. The details of the numerical algorithm used in TLNS3D-MB are available in references 10–11. This code has been calibrated through a wide variety of applications by several independent researchers (refs. 12–15).

For a flow code to be accepted by the research community at large, it must be capable of providing accurate solutions for problems of interest in a timely manner. To a large degree, the accuracy of a flow solver depends

on the explicit and implicit levels of artificial dissipation inherent in the numerical scheme. Based on our earlier research, it was concluded that the accuracy of the central-difference scheme is greatly enhanced by using a matrix type of artificial dissipation model instead of a scalar dissipation model (ref. 16); hence, the matrix dissipation model has been used for the computations presented here. Another key element that determines the overall accuracy of a CFD code is the ability of the underlying turbulence model to capture the pertinent flow physics. If a flow solver has a turbulence model that fails to accurately predict the important flow features, it will not meet the requirements of CFD engineers involved in the design process. For aerodynamic applications, the ability to predict separation zones, shock locations, and boundary-layer properties in the presence of strong pressure gradients is extremely important. In addition, the turbulence model should be able to accommodate multiple surfaces that intersect one another. Based on both a literature survey and our own experience, the one-equation turbulence model of Spalart-Allmaras (ref. 17) can produce accurate solutions for a wide variety of aerodynamics problems, and has been used in the current applications.

The multiblock structure in this code was constructed carefully to minimize the communication lag between the blocks, thereby achieving convergence levels comparable to the single block implementation. Of course, some penalty in the computational efficiency is inevitable because of the overhead associated with the added complexity of coding required in a multiblock code. It is preferred to use the largest block size subject to the geometric constraints to retain the vectorization efficiency. Such a strategy also enhances the implicitness of the numerical scheme by increasing the domain over which the implicit operator of the residual smoothing is effective. These issues are discussed in detail with specific examples by Vatsa, Sanetrik, and Parlette (ref. 10). In the next section, several applications of this method to problems of general interest are discussed.

## Multi-elemen airfoil

During landing and takeoff maneuvers, most aircraft deploy a wing configuration that consists of a multiple number of airfoil sections. For structural and aerodynamic reasons, these components are placed extremely close to one another. Such an arrangement creates special problems in the construction of suitable structured grids, especially $C^0$ continuous grids. An excellent test case representative of realistic high-lift configuration is now available as a result of a joint effort between NASA Langley Research Center and McDonnell Douglas Aircraft Co. for a two–dimensional, 3–element high-lift configuration. The newly available GRIDPRO/AZ3000 software has been used successfully to grid this configuration (ref. 18). A partial view of the resulting 97–block grid is shown here in Fig. 1. The grid lines are smooth and nearly orthogonal, except near singular points formed by the block boundaries. This grid clearly demonstrates the flexibility of GRIDPRO/AZ3000 to generate CFD-quality grids for geometrically complex configurations, and in being able to concentrate grid points near solid surfaces. However, this software package does not lend itself easily to clustering grid points in the field away from the solid surfaces, e.g. along wake lines.

The computed pressure distributions for this configuration were compared with the experimental data by Vatsa et. al in reference 18. In general, the computed pressures agreed well with the measured data, and the resulting solutions indicated consistent treatment of the zonal interface boundary conditions. As expected, a large region of low velocity fluid was observed in the cove regions of the slat and the main airfoil, and on the upper surface of the flap. However, due to poor resolution in the wake regions, the computed velocity profiles did not correlate well with the experimental data (ref. 18).

## F/A-18 forebody/LEX

The next case that was considered for demonstrating the current multiblock code capability was the viscous flow over the F/A-18 forebody leading-edge-extension (LEX) geometry. The test conditions were chosen as $M_\infty = 0.34, Re_{\bar{c}} = 11.5\text{x}10^6$, and $\alpha = 19^0$ to correspond to flight data (ref. 19). In the actual geometry, the LEX on the forebody merges with the wing leading edge. In the simpler model considered here, the grid at the end of

the LEX is extended downstream as a shroud of constant cross-section, which permits the application of a simple extrapolation condition at the downstream boundary. The effect of this simplification should be minimal on the flow over the forebody and the LEX. A similar approach has also been employed by Ghaffari et. al (refs. 20–21). A partial view of the 3–block grid used to model this configuration is shown in Fig. 2. A C-O type grid is used on the forebody (block 1), whereas H-O type grids are employed in the remaining blocks. The block boundaries are selected so that the configuration is subdivided into easily identifiable components.

The computational grid used in this study consists of approximately 750,000 mesh points. A 3.5 order-of-magnitude decrease in the residual of the continuity equation was obtained in about 325 work units (250 fine-grid iterations) in reference 10, which is considered quite good for such a fine mesh, and is significantly better than the convergence rate associated with non-multigrid type of codes. The global force coefficients converged in about half as many iterations. The computed surface-pressure compared favorably with the experimental data and indicated correct trends on both the forebody and the LEX except at the last axial station, where the effect of wing leading-edge missing in the computations becomes significant (ref. 10).

## Subsonic transport aircraft

The next test case that was considered in this study is that of a generic wing/body/engine/pylon configuration. The main reason for selecting this test case is to demonstrate the applicability of the current multiblock code for engine and airframe integration problems, typically encountered in advanced subsonic transport (AST) configurations. Current high-bypass-ratio engines have a very large frontal area and can have a significant effect on the flow field on the wing due to interference effects, which can alter the performance characteristics. Under transonic conditions, these interference effects cannot be predicted accurately with simple linear methods. The particular configuration considered here is a DLR transport aircraft with a high-bypass-ratio engine mounted on a pylon. An 11-block grid consisting of approximately 550,000 mesh points generated by Rossow and Ronzheimer (ref. 22) was used for computing the inviscid flow over this configuration. A partial view of this grid is shown in Fig. 3. The test conditions selected for these computations are $M_\infty = 0.75$ and $\alpha = 0.84°$, which are representative of cruise condition for this type of aircraft. The solutions for this case were reported by Vatsa, Sanetrik, and Parlette in reference 10. The computed surface-pressure contours were found to vary smoothly from one component to the next (ref. 10), which indicates a consistent and accurate treatment across block boundaries, since each component lies in a different block.

To assess the effect of the engine and pylon on the flow over the wing, the surface distributions on two cross sections that lie inboard and outboard of the pylon were compared with the computed pressures on the clean wing and body configuration in reference 10. Based on these comparisons, it was inferred that due to the interference effects caused by the engine and the pylon, the pressure peak flattens in the acceleration region on the lower surface, and the shock shifts forward on the upper surface of the wing, which results in reduced lift. The convergence history for this case was found to be very similar to the F/A-18 case, and a 3.5 order-of-magnitude decrease in the residual was achieved in 400 work units (ref. 10).

## Supersonic transport

Currently there is an enormous interest in United States and elsewhere for developing technology for the next generation of supersonic transports. Because of the high cost of testing a model at flight conditions, the designers must rely heavily on CFD analysis during the developmental phase of such vehicles. A 19-block, $C^0$ continuous structured grid consisting of approximately one million grid points was created to represent a proposed configuration for demonstrating the current capability. A partial view of selected surfaces and zones for this configuration are shown in Figs. 4–6, to indicate the structure and topology of the computational grid. Each component was enclosed within a group of blocks, and the far field was filled with additional blocks. Singular points were introduced at block corners to facilitate an orderly matching of dissimilar topologies.

The Navier-Stokes solutions were obtained for a series of angles of attack at a cruise Mach number of 2.4. A typical run required 3–4 hours of cpu time on a single processor of the NAS Cray C-90 to obtain a converged solutions. The computed forces and moments have been compared with the experimental data, resulting in excellent agreement.

## STATUS OF EMERGING CAPABILITY

The flow solvers on structured grids are in a relatively mature stage of development at this time, and are undergoing mostly incremental changes. On the other hand, grid generation is a rapidly changing field that is experiencing evolutionary changes. In this section, we touch upon the activities in both of these disciplines that in our opinion will have significant impact on future CFD research.

### Grid generation

The biggest bottleneck in the grid-generation process occurs at the domain decomposition level, which is generally a labor-intensive interactive process. In the earlier stages of grid-generation software development, such an approach serves a useful purpose by giving complete control to the user regarding blocking strategies, thereby assisting in the assessment of relative advantages and disadvantages of different strategies. However, for routine grid-generation work, access to batch-oriented grid generation is preferable, in which the desired block structure can be selected via input files, and the grid generation can be completed in an hands-off manner. Two independent developments offer promise regarding domain decomposition. The ICEM/CFD has developed an object-based semi-automated hexahedral volume mesher for creating multiblock structured meshes, known as ICEM/HEXA. The user defines the initial structure or lets HEXA initialize the block structure around a given geometry. Input to the ICEM/HEXA can be either CAD geometry, NURBS surfaces or trimmed NURBS surfaces and NURBS curves. Mesh sizes can be defined on the family of CAD surfaces or individually on the block edges using edge meshing options. The grid is projected onto the underlying CAD geometry with minimum user interaction. The GRIDPRO/AZ3000 software package of Eisemann (ref. 5), on the other hand, employs a different strategy, in which the user specifies the block structure in a topological parametric space. The block boundaries in physical space evolve along with the solution to the grid coordinates. However, a few areas must be improved for this software to provide good quality CFD grids. First, the surface definition within AZ3000 needs a better representation than the bilinear patching that is currently implemented. A more flexible control of grid density is also required in predetermined regions in the field away from solid surfaces to allow clustering in high-gradient regions, such as wakes. Finally, a user-friendly input stream would greatly enhance the usability of this package.

A novel grid-generation methodology has been recently developed by a team of researchers from NASA Langley Research Center and the University of Leeds, England. In this method, known as Rapid Airplane Parametric Input Design (RAPID), a small set of design parameters and grid parameters govern the grid-generation process. The aircraft components (solid surfaces) are manifested through solution of a fourth-order partial-differential equation subject to Dirichlet and Neumann conditions. Volume grids are obtained through an application of the Control Point Method. This technique has been used to generate CFD-quality grids on airplane like configurations that consist of wing, fuselage, horizontal and vertical tails, and canards (ref. 23). This technique provides a medium level of fidelity in terms of surface representation; hence it is more suitable for preliminary conceptual design studies rather than final, detailed analyses.

Although truly automatic hands-off grid-generation capability for complete airplane configurations is still a dream, some recent developments could prove very helpful in parametric design studies. In many applications, CFD engineers are required to study the effect of small geometric changes on the aerodynamic performance of a configuration. The recently developed software package, known as the Coordinate and Sensitivity Calculator for Multidisciplinary Design Optimization (CSCMDO), can be used to generate volumetric grids which reflect small geometric changes in a configuration, given a baseline configuration and the grid associated with it (ref. 24).

This software is controlled via an ASCII user input file for execution in a batch environment. Once the grid on the baseline configuration is available from an independent source, CSCMDO provides the user with a simple, efficient tool for generating grids on perturbed configurations, that are encountered routinely in design optimization of specific class of aircraft.

## Flow solvers

Compared with grid-generation codes, the flow solvers for RANS are at a relatively mature stage in the development cycle. The state-of-the-art flow solvers can provide the steady-state solution to many problems of aerodynamic interest within a day on modern supercomputers. As discussed earlier, most of these flow codes can use $C^0$ continuous grids, and many others can accommodate overlapped and patched grids. However, most of the solvers currently in use are non-conservative on patched boundaries and overlapped zones. Recently, a procedure has become available that can provide us with geometrically conservative interpolation coefficients for patched interfaces on multiblock grids (ref. 25). Work is in progress at NASA Langley Research Center to utilize these interpolation coefficients for developing conservative patched-grid flow solvers.

One of the major hurdles in delivering the latest flow-solver technology into the hands of industrial customers comes from lack of common standards for flow codes. The problem is more acute for multiblock structured codes (as opposed to single-block codes) due to the increased amount of zonal interface information required in such codes. For example, the time required to transfer a NASA-developed code in Boeing's project groups has been estimated by Boeing Engineers to be in years. In recognition of this difficulty, a NASA/Industry team consisting of participants from Boeing, McDonnell Douglas, NASA Ames, Langley and Lewis has been formed to alleviate this problem. The main objective of this team is to develop a Complex Geometry Navier-Stokes Analysis System (CGNS), which would standardize the input/output interfaces for major CFD codes under development at NASA and in the U.S. Aerospace industry. Currently, the team is finalizing the intellectual contents for the CGNS, and a prototype of the system is expected to become available before the end of this year. After the CGNS is developed and has been accepted by the CFD user community, the task of interchanging the flow solvers in a design system will become seamless and straightforward.

# CONCLUSIONS AND RECOMMENDATIONS

Block-structured-grid-based methods offer a viable choice for solving viscous flows over complex aerodynamic configurations. Boundary-fitted structured grids are well suited for resolving the thin viscous layers developing in the vicinity of solid surfaces at high Reynolds numbers typically encountered in flight. State-of-the-art structured flow solvers are known to be very efficient in computing aerodynamic flows in the presence of such embedded boundary layers.

The major stumbling block in routine application of structured-grid methodology to CFD applications is the grid-generation process. Progress is being made to make the surface modeling, domain decomposition, and volumetric grid-generation codes simpler to use. The block-boundaries that are kept fixed in physical space, can severely limit the overall smoothness and orthogonality of the resulting grids. Grid quality can be further enhanced by relaxing the requirement of $C^0$ continuity across block interfaces, i.e. by permitting patched grids at the interfaces. It is much easier to control orthogonality and smoothness across blocks in patched grids. The effort and time required to generate such grids is less compared with equivalent $C^0$ continuous grids.

The availability of different types of blocking strategies would be extremely helpful in the grid-generation packages of future. Both $C^0$ continuous grids and patched grids should be available within the same package. Of course, the use of parametric surface definitions for the underlying geometry will retain the surface fidelity, and the resulting surfaces will correspond more closely with the parametric models from the original CAD definition. Another desirable feature is the ability to accommodate singular points at block boundaries for smoother merging of different topologies (e.g. H-H and C-O).

Future work should continue toward the development of batch-oriented grid-generation codes that require little or no human intervention, after the surface definition has been provided. Since domain decomposition can hinder the automation of block-structured grid generation, effort should focus on automating or simplifying the domain decomposition process. A knowledge-based system, which makes use of grid topologies commensurate with the geometric configurations will be extremely helpful in making this technology more attractive to CFD user community. In addition, grid adaptation strategies should be explored to make more efficient use of grid points.

## ACKNOWLEDGMENTS

The authors are deeply indebted to Dr. S.K. Agrawal of McDonnell Douglas Aircraft Co. for providing the surface grid for F/A-18 configuration and to Dr. C.C. Rossow of DLR (Germany) for providing the grid for subsonic aircraft used in this paper. The authors wish to acknowledge Mary Adams of NASA Langley Research Center for her invaluable help in creating the graphics for the supersonic transport configuration.

## REFERENCES

1. Slotnick, J.P., Kandula,M.; and Buning, P.G.: Navier-Stokes Simulation of the Space Shuttle Launch Vehicle Flight Transonic Flowfield Using a Large Scale Chimera Grid System, AIAA Paper 94–1860, June 1994.
2. Weatherill, N.P.; Eiseman, P.R.; Hauser, J.; and Thompson, J.F. (eds.): Numerical Grid Generation in Computational Fluid Dynamics and Related Fields, Proceedings of the 4th International Conference, Swansea, Wales, Pineridge Press, 1994.
3. Arcill, A.S..; Hauser, J.; Eiseman, P.R.; and Thompson, J.F. (eds.): Numerical Grid Generation in Computational Fluid Dynamics and Related Fields, Proceedings of the 3rd International Conference, Barcelona, Spain, Elsevier Science Publishers B.V., 1991.
4. Smith, R.E. (ed.): Software Systems for Surface Modeling and Grid Generation, NASA CP–3143, Proceedings of a workshop held at NASA Langley, Hampton, VA, 1992.
5. Eiseman, P.: Multiblock Grid Generation with Automatic Zoning, Paper presented at the NASA Workshop on Surface Modeling, Grid Generation, and Related Issues in CFD Solutions, 1995.
6. Rai, M.M.: An Implicit, Conservative, Zonal-Boundary Scheme for Euler Equation Calculations, Computers and Fluids, Vol. 14, No. 3, 1986, pp. 295–319.
7. Klopfer, G.H.; and Molvik, G.A.: Conservative Multizonal Interface Algorithm for the 3–D Navier-Stokes Equations, AIAA Paper 91–1601–CP, 1991.
8. Walters, R.W.; Thomas, J.L.; and Switzer, G.F.: Aspects and Applications of Patched Grid Calculations, AIAA Paper 86–1063, May 1986.
9. Volpe, G.: Conservation and Transparency at Internal Boundaries of Patched Grids, AIAA Paper 93–2930, July 1993.
10. Vatsa, V.N.; Sanetrik, M.D.; and Parlette, E.B.: Development of a Flexible and Efficient Multigrid-Based Multiblock Flow Solver, AIAA Paper 93–0677, Jan. 1993.
11. Vatsa, V.N. and Wedan, B.W.: Development of an Efficient Multigrid Code for 3–D Navier-Stokes Equations and its Application to a Grid-Refinement Study, Computers and Fluids, Vol. 18, No. 4, 1990, pp. 391–403.
12. Yu, N.J.; Allmaras, S.R.; and Moschetti, K.G.: Navier-Stokes Calculations for Attached and Separated Flows Using Different Turbulence Methods, AIAA Paper 91–1791, June 1991.
13. Agrawal, S.; Vatsa, V.N.; and Kinard, T.A.: Transonic Navier-Stokes Flow Computations over Wing-Fuselage Geometries, AIAA Paper 91–3205, Sept. 1991.
14. Marconi, F.; Siclari, M.; Carpenter, G.; and Chow, R.: Comparison of TLNS3D Computations with Test Data for a Transport Wing/Simple Body Configuration, AIAA Paper 94–2237, June 1994.
15. Milholen, W.E.; and Chokani, N.: Low Mach Number Performance of Three-Dimensional Compressible Navier-Stokes Codes, AIAA Paper 95–0767, Jan. 1995.

16. Turkel, E. and Vatsa, V.N.: Effect of Artificial Viscosity on Three Dimensional Flow Solutions, AIAA Journal, Vol. 32, No. 1, Jan. 1994, pp. 39–45.
17. Spalart, P.R.; and Allmaras. S.R.: A One-Equation Turbulence Model for Aerodynamic Flows, AIAA Paper 92–0439, Jan. 1992.
18. Vatsa, V.N.; Sanetrik, M.D.; Parlette, E.B.; Eiseman, P.; and Cheng, Z.: Multiblock Structured Grid Approach for Solving Flows over Complex Aerodynamic Configurations, AIAA Paper 94–0655, Jan. 1994.
19. Fisher, D.F.; Banks, D.W.; and Richwine, D.M.: F-18 High Alpha Research Vehicle Surface Pressures: Initial In-Flight Results and Correlation with Flow Visualization and Wind-Tunnel Data, NASA TM–101724, August 1990.
20. Ghaffari, F.; Luckring, J.M.; Thomas, J.L.; and Bates, B.L.: Navier-Stokes Solutions about the F/A-18 Forebody-Leading-Edge Extension Configuration, Journal of Aircraft, Vol. 27, Sept. 1990, pp. 737–748.
21. Ghaffari, F.: Navier-Stokes, Flight, and Wind Tunnel Flow Analysis for the F/A-18 Aircraft, NASA TP–3478, Dec. 1994.
22. Rossow, C.C. and Ronzheimer, A.: Multiblock Grid Generation Around Wing-Body-Engine-Pylon Configurations, Paper presented at the Third International Conference on Numerical Grid Generation in Computational Fluid Dynamics and Related Fields, Barcelona, Spain, June 1991.
23. Smith, R.E.; Bloor, M.G.I.; Wilson, M.J.; and Thomas, A.M.: Rapid Airplane Parametric Input Design (RAPID), AIAA Paper 95–1687, June 1995.
24. Jones, W.T. and Abolhassani, J.S.: A Grid Generation System for Multi-Disciplinary Design Optimization, Paper presented at the NASA Workshop on Surface Modeling, Grid Generation, and Related Issues in CFD Solutions, 1995.
25. Pao, S.P. and Hamid, A.: General Conservative Patch Algorithm for Multiblock Connectivity, NASA TP (under preparation).

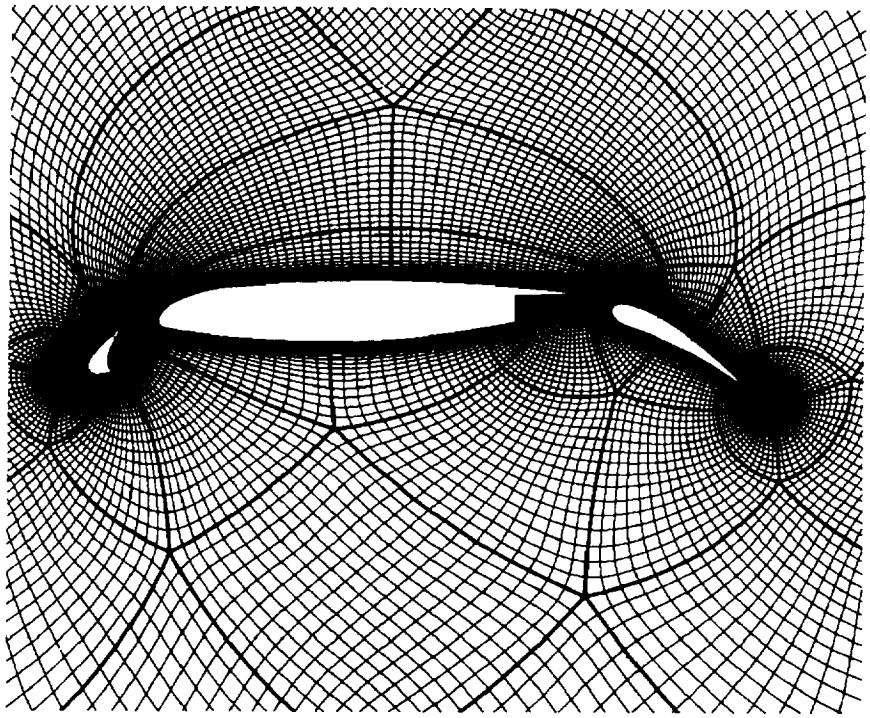


Figure 1.—Partial view of grid for 97–block multielement airfoil configuration.

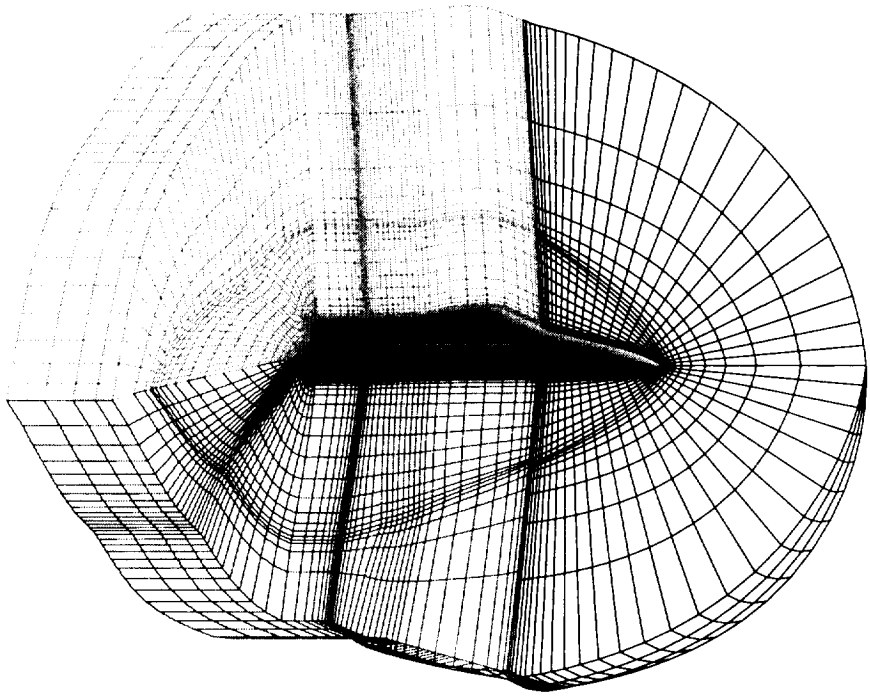


Figure 2.—Partial view of grid for 3–block F/A-18 forebody/LEX configuration.

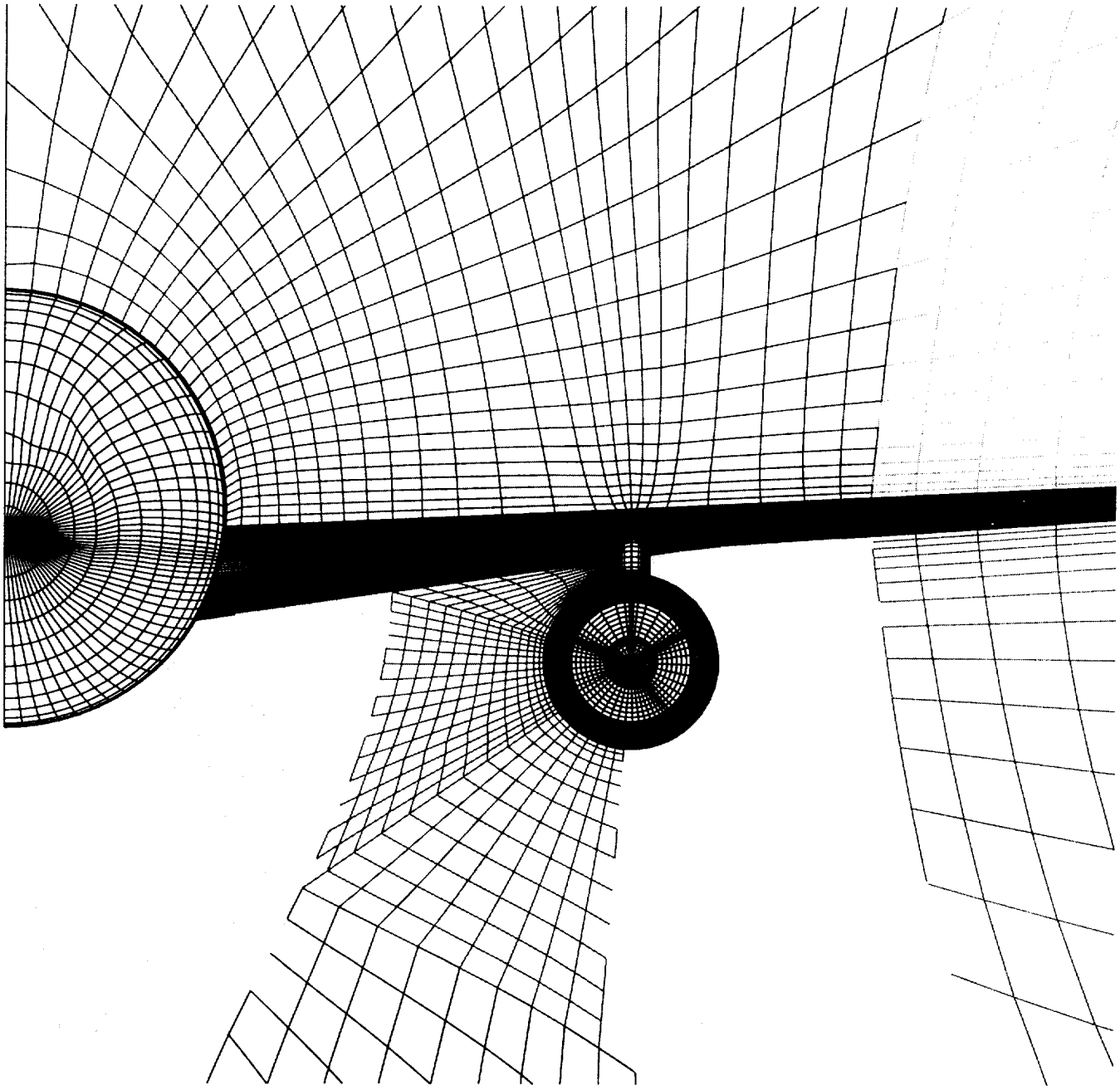


Figure 3.—Surface grid and streamwise cut for subsonic transport configuration.

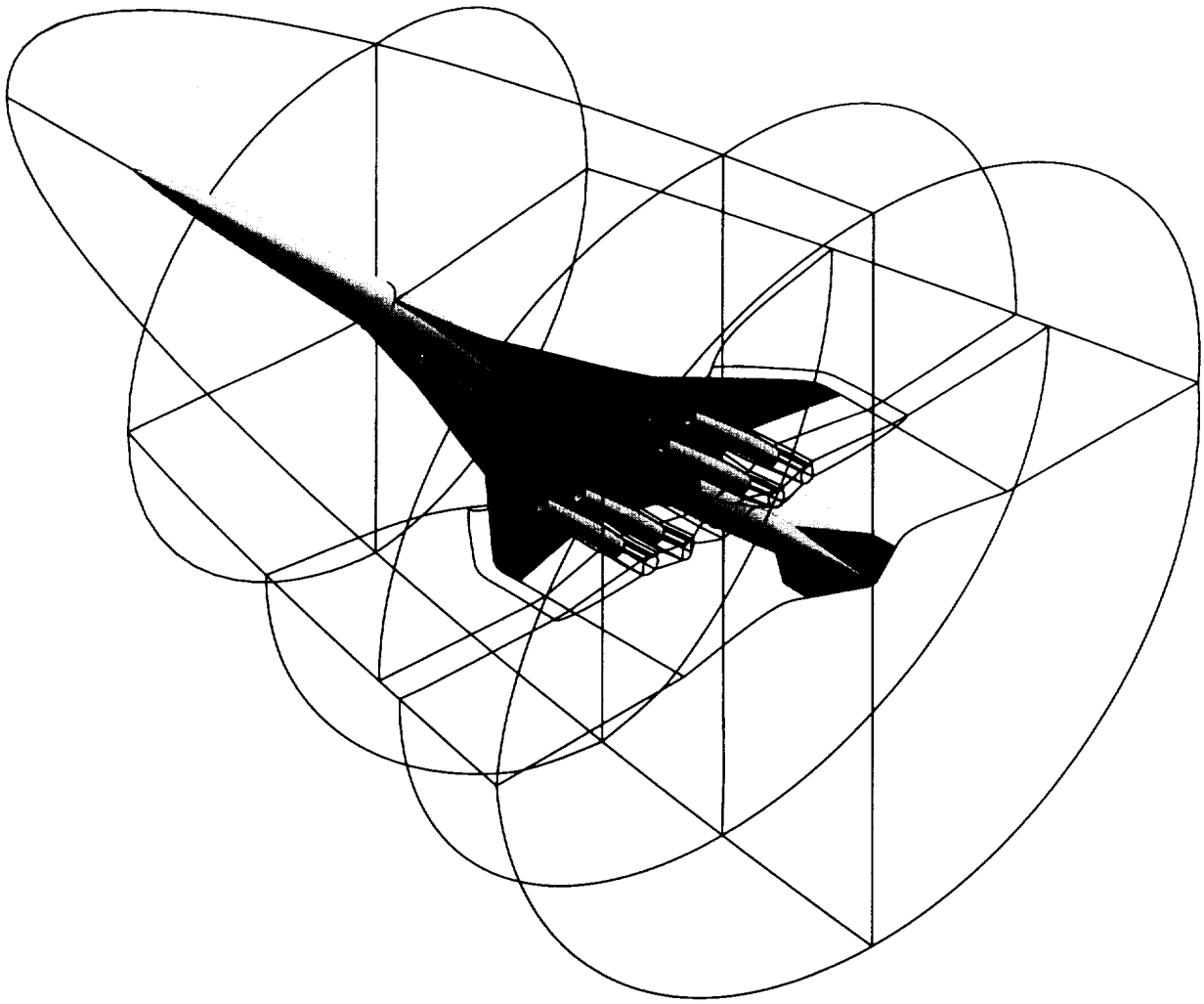


Figure 4.—Grid topology for 19-block supersonic transport configuration.

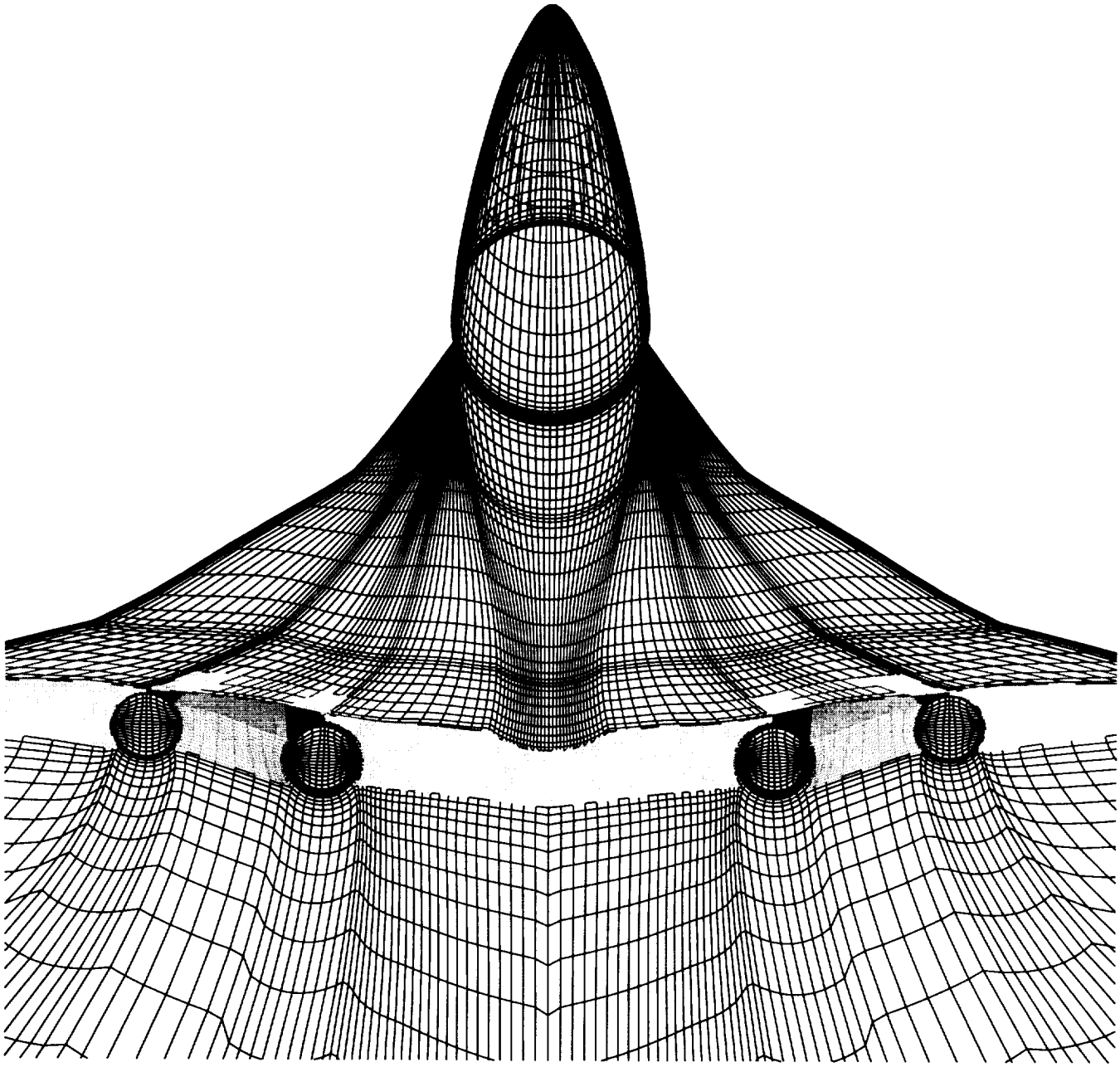


Figure 5.—Surface grid and streamwise cut for wing/fuselage/nacelle/diverter configuration.

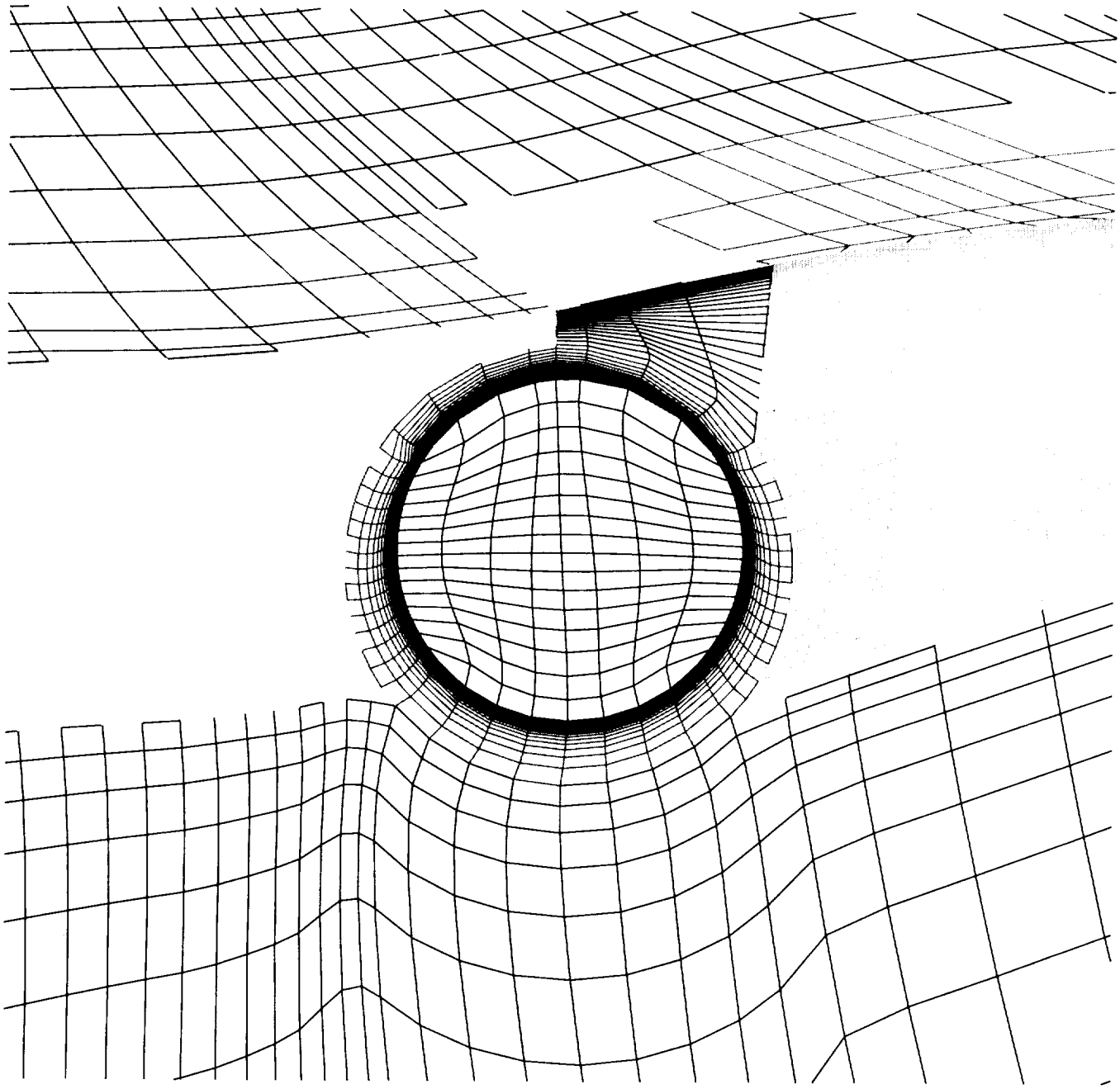


Figure 6.—Detail of nacelle/diverter topology.

# STRUCTURED OVERSET GRID TECHNOLOGY



PRECEDING PAGE BLANK NOT FILMED

N95-28737

19951223316

# GRID RELATED ISSUES FOR STATIC AND DYNAMIC GEOMETRY PROBLEMS USING SYSTEMS OF OVERSET STRUCTURED GRIDS

Robert L. Meakin†
Overset Methods, Inc.
at NASA Ames Research Center M/S 258-1
Moffett Field, CA 94035

## SUMMARY

Grid related issues of the Chimera overset grid method are discussed in the context of a method of solution and analysis of unsteady three-dimensional viscous flows. The state of maturity of the various pieces of support software required to use the approach is considered. Current limitations of the approach are identified.

## INTRODUCTION

Unsteady three-dimensional viscous flow represents an important class of problems for which accurate methods of prediction are frequently required. Such applications are almost always complicated geometrically, may also involve relative motion between component parts, and exist in virtually all engineering disciplines. Experimental methods of analysis, including scale-model and full-scale prototype testing, are often not possible due to excessive cost, model limitations, human safety factors, and time-constraints associated with a commercially competitive environment. Mature computational methods are not always appropriate due to inherent method limitations. Unsteady viscous flowfields involving vortical wakes, interference effects, moving shocks, and body motion demand the most advanced computational means available.

Currently, the only viable high-order method of prediction for these problems is the so called Chimera (ref. 1) overset grid approach. The approach involves the decomposition of problem geometry into a number of geometrically simple overlapping component grids. Multiple-body applications, such as aircraft store-separation (refs. 2-7), are treated naturally in this way. Components of a particular configuration can be altered, or changed completely, without affecting the rest of the grid system (ref. 8). Grid components associated with moving bodies move with the bodies without stretching or distorting the grid system. The approach is applicable to both internal and external flow applications, though most of the Chimera-related algorithm development has thus far been motivated by external flow applications.

The computational incentives for employing an overset grid approach for unsteady three-dimensional viscous flows are multiple. The flow solution process is applied to topologically simple component grids. Body-fitted component grids are ideally suited to regions of thin shear flows such as viscous boundary-layers, wakes, etc. All the advantages associated with structured data are realizable in the approach, including highly efficient implicit flow solvers, memory requirements, vectorization, and fine-grained parallelism. Grid components can be arbitrarily split to optimize the use of available memory resources. Overset structured grid components provide a natural coarse-grained level of parallelism that can easily be exploited to facilitate simulations within distributed computing environments (refs. 9-12).

The present paper considers the current status of the Chimera-style overset grid method as it applies to unsteady three-dimensional viscous flow. Of course, much of what can be said of Chimera in this context is also true for steady-state (viscous and inviscid) applications. The paper includes discussion on the state of maturity of

† *Staff Scientist*

the various pieces of support software required to use the approach, including grid, and flow-solver related issues. Current limitations of the approach are identified and used to suggest needs for future developmental efforts.

## THE CHIMERA OVERSET GRID APPROACH

### Background

In a Chimera-style overset grid approach, domain connectivity is achieved through interpolation of necessary intergrid boundary information from solutions in the overlap region of neighboring grid systems. Consider, for example, the simple two grid discretization of the airfoil shown in figure 1.

The example problem domain is decomposed into a body-fitted grid system near the airfoil surface and a background Cartesian grid system which extends out to the far-field boundaries. The Cartesian grid completely overlaps the airfoil grid. Clearly, the airfoil grid outer boundary conditions can be interpolated from a solution in the off-body Cartesian grid, thereby providing the needed off-body to near-body connection for solution information transfer. It is also clear that a similar transfer of information from the near-body solution back to the off-body solution is required. However, the off-body Cartesian grid has no natural boundaries (physical or numerical) that overlap the near-body grid. The Chimera style of overset gridding makes it possible to create an artificial boundary (hole boundary) within the off-body grid system, and thereby establish the required near-body to off-body connectivity.

A hole boundary for this example is created by excluding the region of the off-body Cartesian grid that is overlapped by the airfoil. The resulting hole region is excluded from the remaining off-body solution. Conditions for the hole boundary are interpolated from the solution in the near-body airfoil grid. In general, one-way communication connections can be established between any set of component grids through hole and outer boundaries. Generalized algorithms for carrying out this task automatically have been developed (refs. 13-19).

### Grid Related Issues

Surface decomposition and surface grid generation represent the primary impediments to the maturation of overset grid based methods. The amount of human resources, measured in time and expertise, currently required to generate suitable systems of overset grids for complex configurations lends validity to the notion that the approach is only an intermediate option, and that unstructured grid approaches will ultimately represent the method of choice for this class of problems. Even if this scenario becomes real, it is currently based on the false assumption that grid generation for structured overset grids is a mature discipline. It also greatly devalues the numerous computational advantages realizable through the use of structured data.

The current difficulties associated with surface decomposition and surface grid generation for overset grid systems exist for the simple reason that there has been virtually no research directed at this area. Available structured grid generation software has been developed almost exclusively for "patched," or "blocked" systems (refs. 20-23) which require neighboring grid components to share a common surface. Although differences between overset and blocked methods may appear slight (i.e., one requires neighboring grids to overlap and the other doesn't), the differences are in fact profound. An overset grid approach is really an unstructured collection of overlapping structured grid components. As such, the approach should enjoy most of the grid generation freedoms associated with unstructured grids, and retain, on a component-wise basis, all of the computational advantages inherent to structured data.

Surface Geometry Decomposition. A good philosophy for a surface geometry decomposition software

package might be to use the fact that all real objects can be viewed as composites of point and line discontinuities, and simple surfaces. For example, the sharp tip of a nose-cone would be a point. Likewise, sharp edges along a fuselage, the trailing edge of a wing, etc. are lines. All object areas that are not associated with points or lines are simple surfaces. Whether an object area corresponds to a point, line, or simple surface dictates the type of surface grid, and hence, volume grid topology that should be used for the overset grid discretization. A point suggests the need for a "nipple" topology for the object area in the vicinity of the point, and an axis topology for the resulting volume grid. A line suggests the need for grid clustering near the line to maintain the integrity of the line discontinuity in the overset grid component. Surface grids associated with line discontinuities lend themselves directly to hyperbolic surface grid generation techniques. Simple surfaces are amenable to either algebraic or elliptic surface grid generation methods. In an overset grid approach, surface grids associated with all object types (i.e., points, lines, and simple surfaces), are amenable to volume grid generation via hyperbolic methods.

Figure 2a illustrates a panel definition of a tiltrotor surface geometry. Object point and line discontinuities are indicated. A point discontinuity exists at the tip of the nose-mounted pitot tube. Line discontinuities exist at the wing/fuselage intersection, wing trailing edge, nacelle exhaust exit, and along the fuselage/sponson crease. One possible surface decomposition of this geometry definition is shown in figure 2b, where no point discontinuities were retained (pitot-tube and mount were neglected at the discretion of the analyst), but line discontinuities were resolved around the wing/fuselage intersection, wing trailing edge, and fuselage/sponson crease. The line discontinuity at the nacelle exhaust exit was smoothed over (at the discretion of the analyst) and treated as a simple surface. As illustrated by figure 2b, the surface grids that result from this method of surface geometry decomposition is a quilt of overlapping surface components.

Surface Grid Generation. Given a suitable surface geometry decomposition, generation of a corresponding set of overset surface grid components should be realizable in a highly automated way. Most of the basic algorithms needed to develop such software currently exist. Algebraic and elliptic surface grid generation techniques, appropriate for simple surfaces, have long been available (ref. 24). The idea for hyperbolic surface grid generation was first put forward more recently (ref. 25), and has since been generalized (ref. 26).

Volume Grid Generation. Generation of volume grids associated with body surfaces can easily be generated in an overset grid approach using hyperbolic grid generation techniques. Hyperbolic volume grid generators exist that are robust, highly efficient, and very easy to use (ref. 27,28). In an overset grid approach, generation of off-body volume grids is a trivial task. The near-body set of grid components must simply be overset onto a convenient background system of grids. While few software packages are currently available to perform the task of off-body grid generation automatically, the task is still trivial and some software is becoming available (ref. 29).

Domain Connectivity. A considerable amount of research and development in the area of domain connectivity among systems of overset grids has been carried out. Several general purpose algorithms for performing this task automatically are currently available. Although existing domain connectivity algorithms can still be improved in terms of efficiency and automation, this area of overset grid technology is maturing rapidly. Active areas of domain connectivity research include Chimera-style hole-cutting (refs. 17-19), donor search methods (including quality optimization) (refs. 14,17,19), automation (refs. 14,17-19), and parallelization (ref. 10).

The first general purpose domain connectivity algorithms that became widely available are the PEGSUS (ref. 13) and, later, CMPGRD (ref. 14) codes. Both codes enjoy substantial use among overset grid practitioners. Likewise, algorithm development associated with both codes is ongoing. In 1989 the first simulations of unsteady three dimensional viscous flow applications involving moving bodies (ref. 3) were carried out using a script controlled application of PEGSUS and the F3D thin-layer Navier-Stokes solver (ref. 30). The need for greater computational efficiency to carry out such applications, which require domain connectivity every time-step, spawned development of alternative domain connectivity algorithms. The DCF3D (ref. 15) and BEGGAR (ref. 17) codes were designed to accommodate moving body applications and are currently the only domain connectivity algorithms that are fully integrated with general purpose flow-solvers and body dynamics algorithms.

## Flow Solver Related Issues

A major advantage of an overset grid approach for solving unsteady three-dimensional viscous flow problems is the fact that existing single grid (structured) flow solvers of documented accuracy and known efficiency can easily be adapted for application within overset grids. For example, the implicit approximately factored algorithm (i.e., block Beam-Warming, ref. 31) for the thin-layer Navier-Stokes equations

$$\partial_\tau \hat{Q} + \partial_\xi \hat{F} + \partial_\eta \hat{G} + \partial_\zeta \hat{H} = R_e^{-1} \partial_\zeta \hat{S} \qquad (1)$$

is easily modified for Chimera-style overset grids as

$$\left[I + i_b \Delta t \delta_\xi \hat{A}^n\right]\left[I + i_b \Delta t \delta_\eta \hat{B}^n\right] \times \qquad (2)$$

$$\left[I + i_b \Delta t \delta_\zeta \hat{C}^n - i_b \Delta t R_e^{-1} \delta_\zeta J^{-1} \hat{M}^n J\right] \Delta \hat{Q}^n =$$

$$- i_b \Delta t \left(\delta_\xi \hat{E}^n + \delta_\eta \hat{F}^n + \delta_\zeta \hat{G}^n - R_e^{-1} \delta_\zeta \hat{S}^n\right)$$

The single and overset grid versions of the algorithm are identical except for the variable $i_b$, which accommodates the possibility of having arbitrary holes in the grid. The array $i_b$ has values of either 0 (for hole points), or 1 (for conventional field points). Accordingly, points inside a hole are not updated (i.e., $\Delta Q = 0$) and the intergrid boundary points are supplied via interpolation from corresponding solutions in the overlap region of neighboring grid systems. By using the $i_b$ array, it is not necessary to provide special branching logic to avoid hole points, and all vector and parallel properties of the basic algorithm remain unchanged.

Solution Accuracy and Conservation. A common criticism of overset grid approaches relates to the fact that simple interpolation is often used to establish needed domain connectivity. Of course, the use of simple interpolation implies that conservation is not strictly enforced. However, assuming the basic flow solver is conservative, conservation is maintained at all points in the domain except at a few intergrid boundary points. The subject of conservation on overlapping systems of grids has been studied by a number of researchers, including (refs. 32-34). In light of the significance typically placed on this subject, several points need to become generally recognized.

First, formal flow solver solution accuracy can be maintained using simple interpolation (refs. 33-35). For example, in a grid refinement study, if the position of component grid outer boundaries remains fixed with increasing resolution of the several grid components, the formal accuracy of a 2nd order flow solver will be maintained with an interpolation scheme that is 2nd order accurate (i.e., tri-linear interpolation of the dependent flow variables).

Second, the primary issue with interpolation of intergrid boundary information is not necessarily one of conservation, but one of grid resolution. If a flow solution is represented smoothly in both donor and recipient grids, simple interpolation is sufficient to carry out simulations that are accurate in all respects. In practical applications, given a fixed number of grid points, it is not possible to provide grid resolution of sufficient density to guarantee that flow features will always be smoothly represented in the grids. If a conservative interpolation scheme is used at intergrid boundaries, the speed and structure of flow features (i.e., shocks, vortices, etc.) may appear continuous across grid interfaces. However, lacking sufficient grid resolution, the accuracy of the solution can not be ensured in any case. Hence, grid resolution is the primary issue.

Third, the objective of adaptive grid techniques is to ensure smooth variation of flow variables throughout the computational domain. Accordingly, an effective adaptive grid technique appropriate for systems of overset grids should be viewed as the primary remedy for issues relating to conservation at grid interfaces.

Finally, methods are available for maintaining conservation at grid interfaces. Although complicated, special interpolation schemes that maintain conservation at grid interfaces have been developed (ref. 36). Interpolation of delta-quantities ($\hat{Q}^{n+1} - \hat{Q}^{n}$), rather than the dependent flow variables ($\hat{Q}^{n+1}$), at grid interfaces has also been suggested as a means for ensuring space-time conservation over the entire domain (ref. 37). Perhaps the most general approach is that of introducing an unstructured grid in the vicinity of the intergrid boundaries (ref. 38) and employing an appropriate solver on the unstructured grid interface. Such a hybrid approach would still have all the advantages of using structured data. Use of an unstructured solver would only be required for a small fraction of the overall domain.

Somewhat less complicated schemes for ensuring conservation at grid interfaces are possible for incompressible flows. Non-conservative interpolation of intergrid boundary conditions can be made conservative by local redistribution of fluxes such that global conservation is ensured (refs. 39,40).

Adaptive Grid Techniques. The subject of adaptive grids has a very large literature. It is clearly not the aim to review this subject here. However, some discussion on the various types of adaption and their respective strengths and weaknesses for application within overset systems of structured grids is provided. The broad class of adaption methods that redistribute a fixed number of points in response to evolving flow features (refs. 41-43) are not considered here. Although such an approach could be implemented within an overset system of grids, there are other methods of adaption that appear to be more general.

Currently, the most popular method of adaption appears to be unstructured cell subdivision. Indeed, the approach is very powerful and general. The approach has been exploited within Cartesian systems, as well as more traditional unstructured grid systems (see figure 3). In either case, the data is unstructured. In the approach, the geometric components of the problem and volume of the domain are discretized with a base grid system. Then, in response to evolving flow features, grid points are added to the base grid by local cell subdivision. Points added to the base grid can be later removed when no longer needed (refs. 44-47). The strength of the approach is that it efficiently allocates grid-points where they are required to maintain solution accuracy. There are several ways in which the approach could be implemented within systems of overset structured grids. The principal drawbacks to the approach are the memory and computational penalties associated with the requisite unstructured data. One possible implementation of this type of solution adaption within an overset structured approach is illustrated in figure 4. The implementation is a hybrid Chimera structured overset grid/unstructured solution adaption algorithm (ref. 49). In the approach, high resolution body-fitted structured grids are used near the bodies (which may move) and are overset onto an unstructured background grid. The bodies cut Chimera holes in the background unstructured grid. All off-body solution adaption is carried out in the unstructured grid using the approach described in (ref. 45).

Another class of solution adaption described in the literature utilizes systems of nested fine overset structured grids. The first such approach suggested the use of nested Cartesian grids (ref. 50) to align with flow features and maintain solution accuracy. Variations of the original approach have continued in the literature and have found application in Cartesian based solution procedures for geometrically complex applications (ref. 51). The basic approach is not limited to Cartesian grids, but can be applied in computational space as well for structured curvilinear grid systems (ref. 52).

A pure Chimera approach to solution adaption has also been explored (refs. 34,53,54). In this approach, structured fine grids are used to resolve flow features with coarse-to-fine and fine-to-coarse grid communication being accomplished via traditional Chimera domain connectivity methods (see figure 5).

Various alternatives to a pure Chimera approach to adaption have also been suggested (refs. 29,54). The approach favored by the author is described in (ref. 29) and divides the solution domain into near-body and off-body regions. Near-body regions of the domain are discretized with high-resolution body-fitted component grids that extend a relatively short distance from body surfaces. The method of adaptive refinement is designed to pro-

vide resolution of off-body dynamics subject to the motion of flow features and/or body components. The off-body portion of the domain is defined to encompass the near-body domain and extend out to the far-field boundaries of the problem. The off-body domain is filled with overlapping uniform Cartesian grids of variable levels of refinement. All adaptive refinement takes place within the off-body component grids. Initially, regions of the off-body field are marked for refinement level based on proximity to near-body boundaries. However, during the solution process, the off-body field is marked for refinement level based on proximity to near-body boundaries and estimates of solution error. Subsequent to refinement level marking, off-body regions of like resolution are coalesced into rectilinear blocks of space, each block becoming a uniform Cartesian grid. Accordingly, at any time during the simulation, the off-body field is discretized with a set of overlapping uniform Cartesian grid systems of varying levels of refinement. The approach is illustrated in figure 6.

The obvious advantages of the overset structured methods of refinement noted above relate to the computational and memory incentives inherent with structured data. The Cartesian based methods noted in (refs. 29,51) offer additional advantages derivable from multiple characteristics of Cartesian systems. For example, no memory is required for grid related data for uniform Cartesian grid components except for the two points that define the diagonal of a box which bounds the grid component and the grid spacing constant. Domain connectivity among systems of uniform Cartesian grids is trivial. Also, highly efficient flow solvers for Navier-Stokes equations on uniform Cartesian grids can be employed.

## CONCLUDING REMARKS

A review of even a small sample of recent applications of overset methods for unsteady three-dimensional viscous flow situations will clearly demonstrate the power and generality of the overall approach. Highly complex geometric configurations can be accurately simulated, including cases involving relative motion between component parts.

All the advantages associated with structured data are realizable in the approach, including highly efficient implicit flow solvers, memory requirements, vectorization, and fine-grained parallelism. Grid components can be arbitrarily split to optimize the use of available memory resources. Decomposition of problem domains into a number of overlapping components creates a coarse-grained level of parallelism that can easily be exploited to facilitate simulations within distributed computing environments.

The subject of surface geometry decomposition for overlapping systems has been heretofore ignored and currently represents the largest impediment to the maturation of Chimera-style overset grid methods. Existing surface grid generation software for blocked, or patched, grid systems do not allow full exploitation of the inherent advantages of overlapping grid systems. Research in this area is badly needed. Other aspects of the Chimera-style overset grid approach are maturing more rapidly. These include algorithm development and generalization for domain connectivity, volume grid generation, surface grid generation, parallel computing, and solution adaptive grid techniques.

## REFERENCES


1. Steger, J.; Dougherty, F. C.; and Benek, J.: A Chimera Grid Scheme, Advances in Grid Generation, K. N. Ghia and U. Ghia, eds., ASME FED-Vol 5., 1983, pp. 59-69.

2. Dougherty, F. C.; Benek, J.; and Steger, J.: On Applications of Chimera Grid Schemes to Store Separation, NASA TM-88193, 1985.

3. Meakin, R. and Suhs, N.: Unsteady Aerodynamic Simulation of Multiple Bodies in Relative Motion, AIAA Paper 89-1996-CP, 9th CFD Conf., 1989, pp. 643-657.

4. Meakin, R.: Computations of the Unsteady Flow About a Generic Wing/Pylon/Finned-Store Configuration, AIAA Paper 92-4568-CP, AFM Conf., 1992, pp. 564-580.

5. Lijewski, L. and Suhs, N.: Time-Accurate Computational Fluid Dynamics Approach to Transonic Store-Separation Trajectory Prediction, AIAA J. Aircraft, vol. 31, no. 4, 1994, pp. 886-891.

6. Atwood, C.: Computation of a Controlled Store Separation from a Cavity, AIAA Paper 94-0031, 32rd ASM&E, 1994, Reno, NV.

7. Sahu, J. and Nietubicz, C.: Application of Chimera Technique to Projectiles in Relative Motion, AIAA Paper 93-3632-CP, AFM Conf., 1994, pp. 167-176.

8. Gea, L., Halsey; N, Intemann, G.; and Buning, P.: Applications of the 3-D Navier-Stokes Code OVERFLOW for Analyzing Propulsion-Airframe Integration Related Issues on Subsonic Transports, ICAS-94-3.7.4, Proc. 19th Congress Intnl. Council Aero. Sci., 1994, pp. 2420-2435.

9. Smith, M. and Pallis, J.: MEDUSA - An Overset Grid Flow Solver for Network-Based Parallel Computer Systems, AIAA Paper 93-3312-CP, 11th CFD Conf., 1993, pp. 167-175.

10. Barszcz, E.; Weeratunga, S.; and Meakin, R.: Dynamic Overset Grid Communication on Distributed Memory Parallel Processors, AIAA Paper 93-3311-CP, 11th CFD Conf., 1993, pp. 155-166.

11. Atwood, C. and Smith, M.: Nonlinear Fluid Computations in a Distributed Environment, AIAA Paper 95-0224, 33rd ASM&E, 1995, Reno, NV.

12. Weeratunga, S. and Chawla, K.: Overset Grid Applications on Distributed Memory MIMD Computers, AIAA Paper 95-0573, 33rd ASM&E, 1995, Reno, NV.

13. Benek, J.; Steger, J.; Dougherty, F.; and Buning, P.: Chimera: A Grid-Embedding Technique, AEDC-TR-85-64, 1986.

14. Brown, D.; Chesshire, G.; Henshaw, W.; and Kreiss, O.: On Composite Overlapping Grids, 7th Intnl. Conf. Finite Element Methods in Flow Probs., 1989, Huntsville, AL.

15. Meakin, R.: A New Method for Establishing Intergrid Communication Among Systems of Overset Grids, AIAA Paper 91-1586-CP, 10th CFD Conf., 1991, pp. 662-671.

16. Suhs, N. and Tramel, R.: PEGSUS 4.0 User's Manual; AEDC-TR-91-8, 1991.

17. Maple, R. and Belk, D.: Automated Set Up of Blocked, Patched, and Embedded Grids in the Beggar Flow Solver, Numerical Grid Generation in Computational Fluid Dynamics and Related Fields, ed. N.P. Weatherill et al., 1994, Pine Ridge Press, pp. 305-314.

18. Wey, T.: Development of A Mesh Interface Generator for Overlapped Structured Grids, AIAA Paper 94-1924, 12th AA Conf., 1994.

19. Chiu, I. and Meakin, R.: On Automating Domain Connectivity for Overset Grids, AIAA Paper 95-0854, 33rd ASM&E, 1995, Reno, NV.

20. Steinbrenner, J.; Chawner, J.; and Fouts, C.: A Structured Approach to Interactive Multiple Block Grid Generation, AGARD FDP Specialists Mtg. on Mesh for Complex Three-Dimensional Configurations, 1989, Leon, Norway.

21. Thompson, J.: Composite Grid Generation Code for General 3-D Regions - the EAGLE Code, AIAA Journal, vol. 26, no. 3, 1988, pp. 271-272.

22. Soni, B., et al.: GENIE++, EAGLEView and TIGER: General and Special Purpose Graphically Interactive Grid Systems,

AIAA Paper 92-0071, 30th ASM&E, 1992, Reno, NV.

23. Akdag, V. and Wulf, A.: Integrated Geometry and Grid Generation System for Complex Configurations, Software Systems for Surface Modeling and Grid Generation, NASA CP 3143, 1992, pp. 161-171.

24. Thompson, J., Warsi, Z., and Mastin, C.: Boundary-Fitted Coordinate Systems for Numerical Solution of Partial Differential Equations - A Review, J. Comput. Phys., vol. 47, no. 1, 1982, pp. 1-108.

25. Steger, J.: Notes on Surface Grid Generation using Hyperbolic Partial Differential Equations, unpublished report, TM CFD/UCD 91-101, Dept. Mech., Aero. & Mat. Eng., 1989, Univ. Cal., Davis.

26. Chan, W. and Buning, P.: A Hyperbolic Surface Grid Generation Scheme and its Applications, AIAA Paper 94-2208, 25th FD Conf., 1994, Colorado Springs, CO.

27. Chan, W. and Steger, J.: A Generalized Scheme for Three-Dimensional Hyperbolic Grid Generation, AIAA Paper 91-1588-CP, 10th CFD Conf., 1991, pp. 683-695.

28. Chan, W., Chiu, I., and Buning, P.: User's Manual for the HYPGEN Hyperbolic Grid Generator and the HGUI Graphical User Interface, NASA TM 108791, 1993.

29. Meakin, R.: An Efficient Means of Adaptive Refinement Within Systems of Overset Grids, AIAA Paper 95-1722-CP, 12th CFD Conf., 1995, San Diego, CA.

30. Steger, J.; Ying, S.; and Schiff, L.: Partially Flux-Split Algorithm for Numerical Simulation of Compressible Inviscid and Viscous Flow, Workshop CFD, Inst. Non-Linear Sci., 1986, Univ. of Cal., Davis.

31. Warming, R. and Beam, R.: On the Construction and Application of Implicit Factored Schemes for Conservation Laws, SIAM-AMS Proc., vol. 11, 1978, pp. 85-129.

32. Berger, M.: On Conservation at Grid Interfaces, SIAM J. Numer. Anal., vol. 24, no. 5, 1987, pp. 967-984.

33. Chesshire, G. and Henshaw, W.: Conservation on Composite Overlapping Grids, IBM Research Report, RC 16531, IBM Res. Div., T. J. Watson Res. Ctr., 1991, Yorktown Hts., NY.

34. Meakin, R.: On the Spatial and Temporal Accuracy of Overset Grid Methods for Moving Body Problems, AIAA Paper 94-1925-CP, 12th AA Conf., 1994, pp. 858-871.

35. Henshaw, W.: Part I: The Numerical Solution of Hyperbolic Systems of Conservation Laws. Part II: Composite Overlapping Grid Techniques, Ph. D. Diss., Cal. Inst. of Tech., 1985, Pasadena, CA.

36. Moon, Y. and Liou, M.: Conservative Treatment of Boundary Interfaces for Overlaid Grids and Muli-level Grid Adaptations, AIAA Paper 89-1980-CP, 9th CFD Conf., 1989, pp. 480-487.

37. Steger, J.: Studies of the Chimera Method of Flow Simulation (unpublished notes), 1991.

38. Wang, Z.: A Fully Conservative Structured/Unstructured Chimera Grid Schem, AIAA Paper 95-0671, 33rd ASM&E, 1995, Reno, NV.

39. Meakin, R.: Application of Boundary Conforming Coordinate and Domain Decomposition Principles to Environmental Flows, Ph. D. Diss., Stanford Univ., 1986, Stanford, CA.

40. Saltzman, J.: Conservation and Linear System Issues on Overset Composite Grids, Proc. 2nd Overset Composite Grid Sol. Tech. Symp., 1994, Fort Walton Beach, FL.

41. Brackbill, J. and Saltzman, J.: Adaptive Zoning for Singular Problems in Two Dimensions, J. Comput. Phys., vol. 46,

1982, pp. 342-368.

42. Eiseman, P.: Adaptive Grid Generation, Comput. Meth. Appl. Mech. Eng., vol. 64, nos. 1-3, 1987, pp. 321-376.

43. Davies, C. and Venkatapathy, E.: Application of a Solution Adaptive Grid Scheme, SAGE, to Complex Three-Dimensional Flows, AIAA Paper 91-1594-CP, 10th CFD Conf., 1991, pp. 756-779.

44. Zeeuw, D. and Powell, K.: An Adaptively-Refined Cartesian Mesh Solver for the Euler Equations, AIAA Paper 91-1542-CP, 10th CFD Conf., 1991, pp. 166-180.

45. Biswas, R. and Strawn, R.: A New Procedure for Dynamic Adaption of Three-Dimensional Unstructured Grids, AIAA Paper 93-0672, 31st ASM&E, 1993, Reno, NV.

46. Aftosmis, M.: Upwind Method for Simulation of Viscous Flow on Adaptively Refined Meshes, AIAA J., vol. 32, no. 2, 1994, pp. 268-277.

47. Baum, J. Luo, H. and Lohner, R.: A New ALE Adaptive Unstructured Methodology for the Simulation of Moving Bodies, AIAA Paper 94-0414, 32nd ASM&E, 1994, Reno, NV.

48. Barth, T.: Aspects of Unstructured Grids and Finite-Volume Solvers for the Euler and Navier-Stokes Equations, AGARD-R-787, 1992, pp. 6.1-6.61.

49. Duque, E.: A Structured/Unstructured Embedded Grid Solver for Helicopter Rotor Flows, AHS 50th Ann. Forum Proc., vol. II, 1994, pp. 1249-1258.

50. Berger, M. and Oliger, J.: Adaptive Mesh Refinement for Hyperbolic Partial Differential Equations, J. Comput. Phys., vol. 53, 1984, pp. 484-512.

51. Berger, M. and LeVeque, R.: A Rotated Difference Scheme for Cartesian Grids in Complex Geometries, AIAA Paper 91-1602, 10th CFD Conf., 1991, Honolulu, HI.

52. Berger, M. and Jameson, A.: Automatic Adaptive Grid Refinement for the Euler Equations, AIAA J., vol. 23, no. 4, 1985, pp. 561-568.

53. Chawla, K. and Banks, D.: Tracking Flow Features Using Overset Grids, AIAA Paper 93-0197, 31st ASM&E, 1993, Reno, NV.

54. Rogers, S. and Pulliam, T.: Accuracy Enhancements for Overset Grids Using a Defect Correction Approach, AIAA Paper 94-0523, 32nd ASM&E, 1994, Reno, NV.

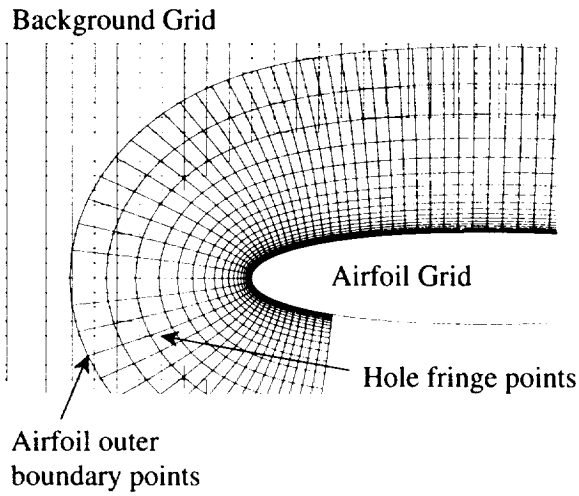
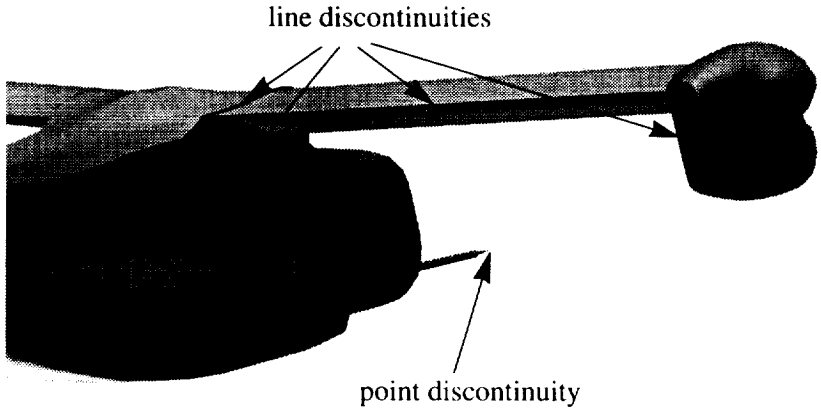
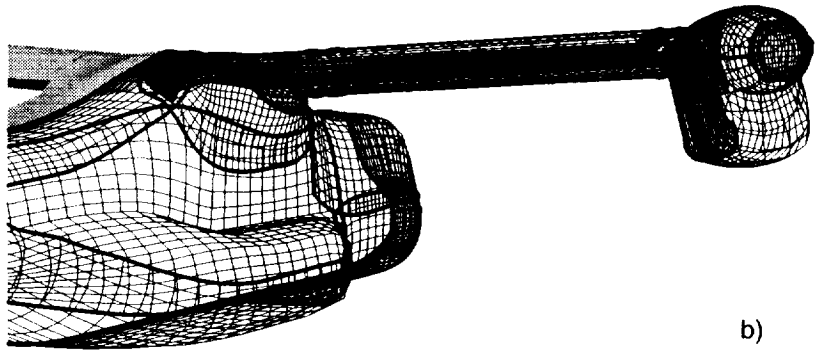



**Figure 1.** —An example of Chimera intergrid boundary points




a)



b)

**Figure 2.** — Surface geometry of a tiltrotor aircraft. a) Panel definition (note indicated point and line discontinuities). b) Quilt of overlapping surface grid components. Component grids retain the line discontinuities indicated in the original panel definition of the geometry.

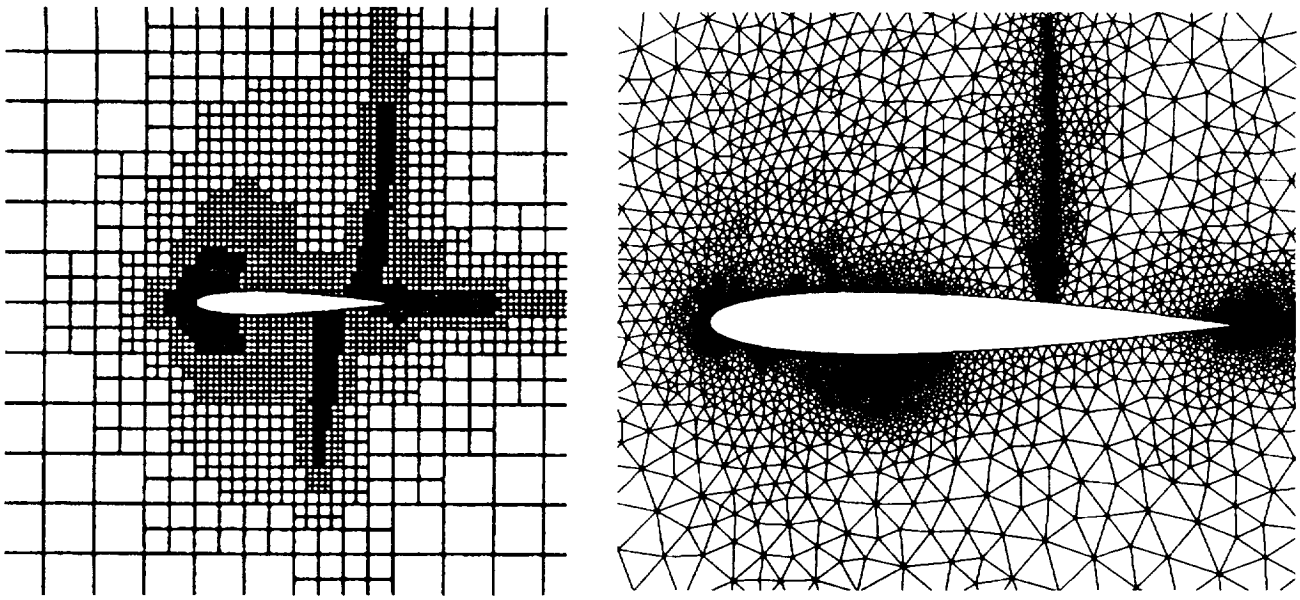



**Figure 3.** — Solution adaption via cell subdivision. a) Unstructured Cartesian cell subdivision (ref. 44). b) Conventional unstructured cell subdivision (ref. 48).

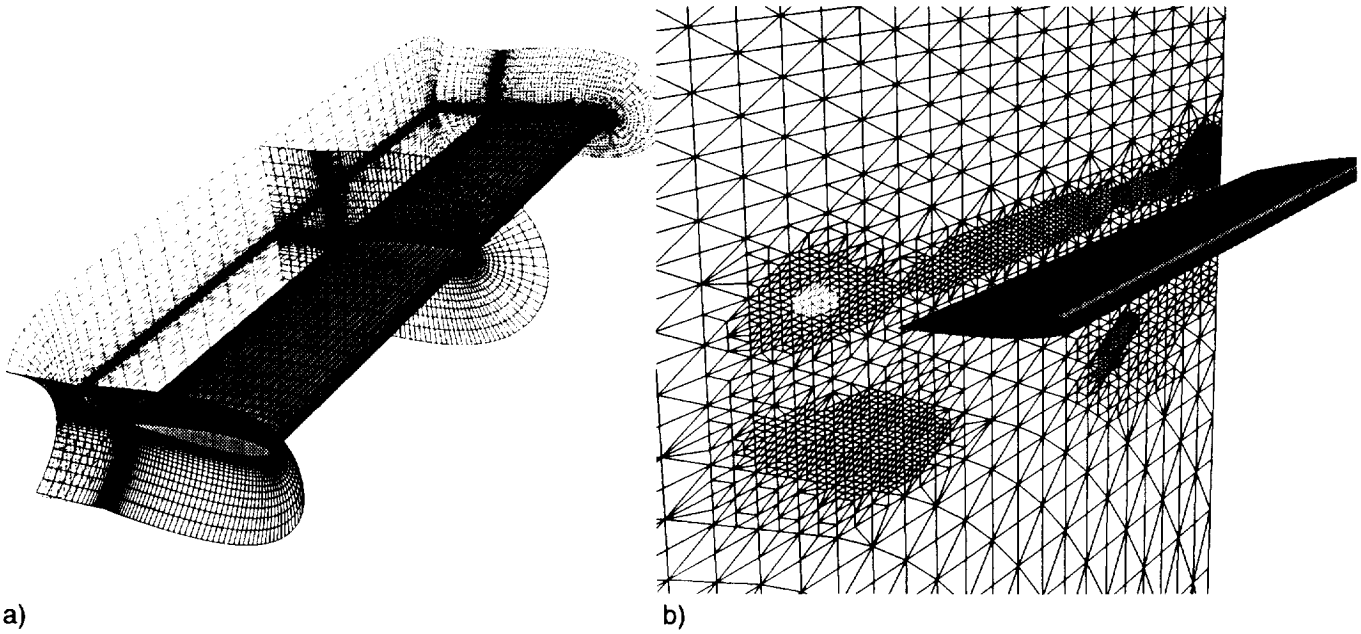



**Figure 4.** — Hybrid structured Chimera overset/unstructured solution adaption. a) High-resolution body-fitted structured grid (viscous) for rotor blade. b) Unstructured background grid for solution adaption of off-blade vortex dynamics (ref. 49).

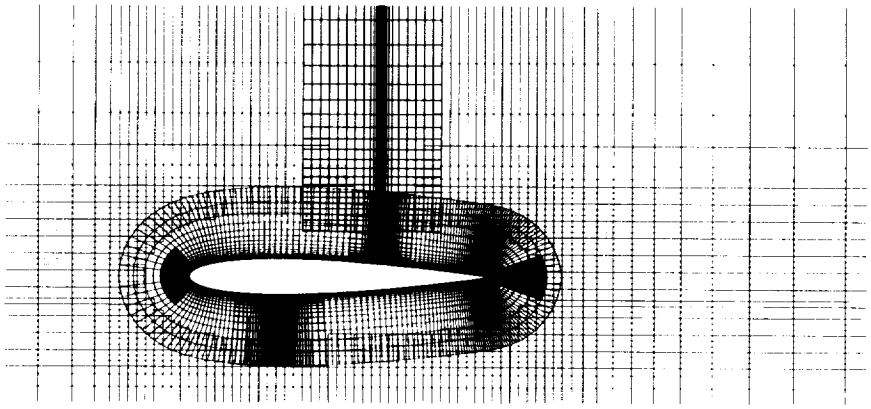
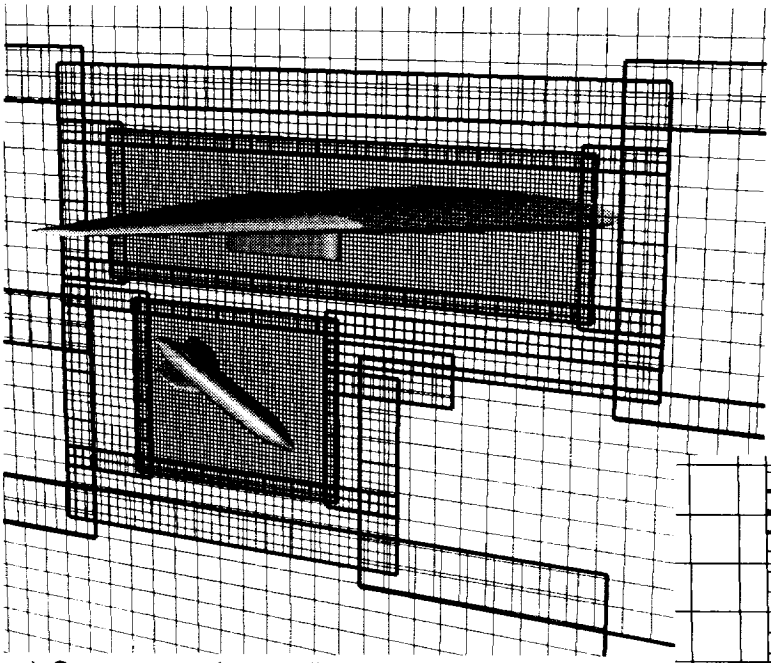
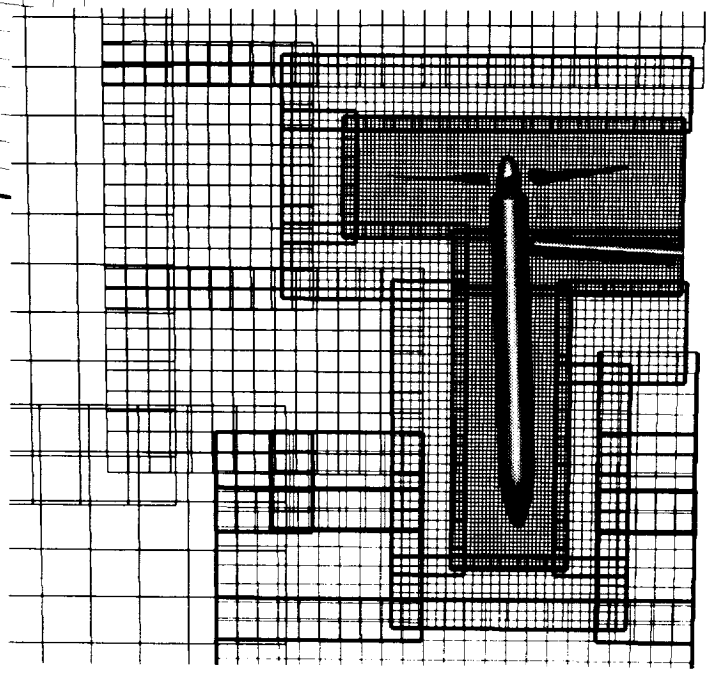


**Figure 5.** — Solution adaption via overset structured fine grids. Base grid (medium resolution body-fitted airfoil grid and background Cartesian grid) plus 5 overset fine grid components (ref. 34).



a) Store separation application (near-body viscous grid components not shown).



b) Tiltrotor downloads application (near-body viscous grid components not shown).

**Figure 6.** — Solution adaption via off-body uniform Cartesian structured grids (high-resolution body-fitted grids are used to discretize space near physical boundaries) (ref. 29).

N95-28738

19951223 17

# THE ROLE OF OVERSET GRIDS IN THE DEVELOPMENT OF THE GENERAL PURPOSE CFD CODE

Davy M. Belk
Air Force Wright Laboratory
Eglin AFB, Florida 32542

## SUMMARY

A discussion of the strengths and weaknesses of overset composite grid and solution technology is given, along with a sampling of current work in the area. Major trends are identified, and the observation is made that generalized and hybridized overset methods provide a natural framework for combining disparate mesh types and physics models. Because of this, the author concludes that overset methods will be the foundation for the general purpose computational fluid dynamics programs of the future.

## INTRODUCTION

Overset grid methods have been used to great advantage in the solution of flow fields over geometrically complex configurations, and for multiple bodies with relative motion. The usefulness of the overset approach has been demonstrated for problems ranging from three-dimensional flow around a ship (see appendix, Malmliden) to prediction of trajectories of aft ejected submunitions in supersonic flow (see appendix, Sahu and Nietubicz) and hypersonic HEDI shroud separation (see appendix, Narain). Other approaches for modeling complex geometry, such as the use of unstructured grids or block structured grids, do not offer the advantage of no regridding when bodies move relative to one another.

Although proven useful, there are definitely areas that need improvement. Among these are the usability of the codes that assemble systems of overset grids, the awkwardness of having multiple solutions in certain flow regions, and concerns over conservation errors at internal boundaries. Fortunately, there is a great deal of current interest in overset methods, and innovative solutions are being tested against these problems. Much of this work was discussed at the "Second Overset Composite Grid and Solution Technology Symposium," held at Fort Walton Beach, Florida in October 1994. The symposium was hosted by the Northwest Florida Section of the American Institute of Aeronautics and Astronautics. At this symposium, both development and applications related to overset grid methods were presented. (This paper will draw heavily on material presented at the symposium. The procedure for obtaining copies of a particular presentation are given in the appendix.) A review of accomplishments and current development in overset grid and solution technology has led the author to the conclusion that overset methods will be widely used in the future. In fact, the assertion is made that overset methods, in generalized and hybridized forms, will be the foundation for the general purpose computational fluid dynamics programs of the future.

In the remainder of this paper, strengths and weaknesses of overset composite grid methods will briefly be discussed, and representative work aimed at remedying the weaknesses will be referenced. Major trends in development will be identified, and the conclusion about the future of overset methods will be stated.

## STATE OF THE ART

### Strengths

The strong points of the overset grid methods are numerous. One of the most important items, and also one of the original motivations for this approach, is the ability to model bodies undergoing relative motion. As long as the bodies are rigid, the mesh system associated with the body can move with it and not require regeneration. This is a significant simplification that makes possible the solution of some problems that otherwise would be intractable. Another advantage is the flexibility afforded by the independent meshing of components. This allows selection of grid topology to suit the local geometry, and reduces the global ramifications of this selection. Flexibility also comes in the form of freedom to choose physics models and computational algorithms differently in the different grids. If the physical problem admits localization of viscous, chemical, or other effects, then a tailored grid and solution procedure can be placed in the locality required.

Overset methods preserve the efficiency of structured grid solvers (also of other solution methods, as will be discussed later). Solution algorithms typically require minimal modification to operate on an overset grid. The logic required to determine holes and interpolation coefficients and other items associated with assembling a system of overset grids is usually divorced from the solver. The solver only needs to accept an array indicating cells which are not to be included in the solution. Canned routines are usually used to perform intergrid communication, letting the solver be "ignorant" of most communication issues.

An alternative adaptive refinement is possible with overset grids. Adaptive refinement can be achieved in overset grid systems by inserting fine grids in regions in need of resolution.

The next round of advances in high performance computing capability is generally expected to come in the form of a shift from vector processing to massively parallel processing. Domain decomposition into overset grids is a natural mapping of existing algorithms to parallel machines. This has been done at Arnold Engineering and Development Center (see Benek, appendix), with quite high efficiency. Jesperson and Levit (see appendix) showed that spreading each grid across processors on a CM5 is also an option. Their overhead for intergrid communication was "noticeable but not insurmountable."

The mode of problem solution using overset grids is well suited to a production environment. The required component grids of a complex geometry can be developed semi-independently by different engineers, and assembled by yet another engineer. A high degree of reuse is possible with grids developed for use in an overset environment, resulting in libraries of frequently used geometries that can be easily put together. For example, if models of the F-16, the 370 gallon tank, the weapons pylon, and several different weapons are sitting on the shelf, then it is a relatively simple procedure to investigate the aerodynamics of the aircraft and various combinations of tank and weapons. If multiple copies of the same weapon are to be carried, then the engineer simply inputs how many and where. If another aircraft model is available, the same weapons can be put on this aircraft very quickly.

It is also relatively easy to add small appendages to previously developed models. Jolly, et al. (see appendix) showed an example of a quick turnaround analysis of an external battery pack addition to a weapon. The previously developed weapon grid was not modified, but served as the outer grid into which the battery pack grid was inserted, as shown in Figure 1.

## Weaknesses

A frank discussion of overset grid and solution methods requires that consideration be given to the valid criticisms of these methods. These areas of concern fall into familiar categories of accuracy, efficiency, usability, and algorithmic complexity. A synopsis of the issues is given below, and a discussion of ongoing research targeting these issues will be given in the next section.

Conservation at internal boundaries is the most often voiced concern about the accuracy (or even the validity) of overset method application to problems involving shocks and discontinuities. Communication between domains via interpolation of boundary values is, in general, nonconservative. Meakin (ref. 1) performed a careful study of spatial and temporal accuracy of overset grid methods through numerical experimentation. He found that if a flow solution is represented smoothly in both donor and recipient grids, simple interpolation is sufficient to maintain formal solution accuracy. Still, this issue continues to be bothersome since practical situations arise where the flow solution is not represented smoothly on both grids. In these cases, according to Meakin, a conservative interface scheme is preferable, but accuracy will be compromised regardless.

The efficiency issue is one of wasted calculations on both regions with multiple overlapping grids and in blanked regions. In overlapping regions, each grid can have a calculated solution, and each solution will be different due to the different discretizations. Cells in blanked regions usually undergo the same set of calculations as field cells in order to maintain vectorization of code. This can add up to a considerable number of wasted calculations, and is particularly wasteful on cache-based (non-vector) machines.

Usability is not at the desired level as yet. Considerable user expertise and interaction is required to determine where to cut holes, from which grids to interpolate values, how to generate the grids to achieve resolution matching in interpolated regions, and how to prevent circular interpolations.

Finally, the computer code required to implement overset methods can become quite complex, especially when an attempt has been made to address some of the previously mentioned issues. What begins as a straightforward idea can quickly become unwieldy in its implementation.

# DEVELOPMENT

The development work described below is aimed at alleviating the existing weaknesses of overset methods. Much of the work described was presented at the 2nd Overset Composite Grid and Solution Technology Symposium (see appendix).

## Conservation and Solution Accuracy

Most of the techniques aimed at ensuring conservation have achieved this result by actually eliminating overlap between the grids in the system. Wang's method (see appendix) in a region of two grid overlap is to leave one grid whole and eliminate the other grid from the overlap region. The precise intersection of the outer boundary of the whole grid with the cut grid is mapped out, and unique fluxes are calculated and distributed appropriately along this boundary. Kao and Liou (ref. 2, also see appendix) use a method they have dubbed the Direct Replacement of Arbitrary Grid-overlapping by Nonstructured (DRAGON) Grid technique. Here, overlapping grids are pared back to eliminate overlap, and glued together with an unstructured mortar grid. Another idea, not so directly motivated by conservation issues as by the advantage of unstructured grids in a

particular region, is to simply tie an embedded unstructured grid to a structured grid. This has been promoted by Wang (see appendix) and others. In this form, there is little conceptual distinction from hybrid schemes that essentially embed a structured grid near no slip boundaries and use unstructured outer grids.

## Multiple Competing Solutions

Wasted Calculations.-- For complex problems, mesh arrangements can easily occur that have multiple layers of grids in some regions of space. Instead of allowing each grid to have its own different solution, and communicating between meshes only on the grid boundaries and hole fringes, Johnson and Belk (ref. 3) use these layers of grids as levels in a nonaligned multigrid scheme. Interpolation of residuals and dependent variables from finer meshes to coarser meshes throughout the region of overlap is used to calculate a defect correction driving the coarse grid solution. The coarse grid solution is then used to update the fine grid variables. This procedure was shown to provide a means for improving accuracy by embedding grids aligned with flow features, but without the need to cut holes. Also, the convergence rate of the resulting grid system was significantly enhanced in comparison to the standard overset communication scheme of cutting holes and interpolating at boundaries.

In a similar manner, Rogers and Pulliam (ref. 4) demonstrated the advantages of a defect correction approach for communication between overset grids. In this work, the simplification of not actually calculating a fine grid solution was used. Instead, the coarse grid solution is interpolated to the finer embedded grid and a defect correction is calculated to drive the coarse grid solution to higher accuracy.

One way to minimize wasteful calculations is to blank out as many overlapping cells as possible. (Of course, unless the code uses conditional execution to skip blanked cells, this has negligible influence on the number of floating point operations required for a solution step.) The grid assembly algorithm of Chesshire and Henshaw (ref. 5, also see appendix) has the property of minimizing overlap subject to certain criterion on relative cell sizes. Several others, such as Wey (ref. 6, also appendix) and Chiu and Meakin (see appendix) have techniques to minimize overlap. In these methods, the hole boundary expands until it can go no further and still maintain valid overset communication.

Still another approach is to eliminate overlap entirely, such as the schemes discussed earlier under the topic of conservation.

Force Integration. -- A practical problem that results from the existence of multiple overlapping solutions is force integration on bodies. For example, a fin added to a missile using overset methods will result in a region near the fin with both the fin grid and the body grid conforming to the missile body. There will also be a region of invalid body cells, or holes, adjacent to the fin, as shown in Figure 2. Accurate integration of predicted pressure and shear stress to obtain body forces and moments requires some scheme to prevent either doubly counting some regions, or leaving holes.

One option is to ignore the problem. If the mesh is fine, and the hole in the body grid near the fin is relatively small, then the body force contribution can be adequately represented by using the body grid alone. While not the most satisfying solution, the results of Lijewski and Suhs (ref. 7) show that trajectories of separating stores can be accurately predicted with this approximation.

Another approach is that of Dietz, who wrote the TESS code to handle this situation (see appendix). The TESS code includes all points from overlapping grids on a surface, and triangulates the intermixed points to

produce a unified surface mesh. The points carry their properties with them, and the resulting triangular mesh is used to integrate forces and moments.

Chan's method (ref. 8, also see appendix), on the other hand, first eliminates overlap on the surface by removing points belonging to the coarsest overlapping grid. The pared back surface meshes are then reconnected with a triangular grid, and the forces are integrated on this mesh.

The author's approach (ref. 9, also see appendix) is to try to avoid this problem by representing static assemblages of components with block structured grids called superblocks. Within a superblock, no grid overlapping occurs, except implicitly at boundaries. The superblocks are allowed to be placed in arbitrary overset arrangements. Figure 3 shows a wing, pylon, and finned store grid system with a blocked-grid superblock around the store.

## Usability

Significant improvements in usability of overset grids and solvers have come from attempts to automate the grid assembly process. Interactive tools have also been developed to aid in the grid assembly process. One of the most important developments from a user's standpoint, however, is the work of Chan et al. (ref. 8, 10, also see appendix) to develop grid generation tools for overset systems. Most of the existing overset grid assembly tools still require externally generated grids. If the grids are not compatible in terms of having comparable point density and a sufficient number of cells to allow an interpolated fringe after hole cutting, then the meshes must be modified. In addition, grids around intersecting surfaces must either be built to conform to both surfaces, or a collar grid must be used to avoid a region where points from all grids have been blanked. Chan's SURGRD code (ref. 8) is a surface grid generator tailored for overset grids, and the HYPGEN (ref. 10) code is a volume grid generation code particularly useful for overset grids.

Interactive tools are available to set up the input for Meakin's DCF3D Code. These tools simplify the specification of analytic hole cutting surfaces, and give immediate feedback on the relative location of the hole boundaries and bodies. Search hierarchy, boundary information, and other information is set interactively.

Easy assembly of overset grids has always been a goal. However, different approaches have been taken concerning the degree of user control desired or required in the process. The most mature code that has attempted to minimize user control requirements is Chesshire and Henshaw's CMPGRD program (see appendix). The code uses an algorithm that determines hole size and overset connections with little user intervention, attempting to minimize resolution mismatch in interpolated regions. CMPGRD is also set up to allow for multigrid within each component grid. Chiu and Meakin presented work towards automating domain connectivity for overset grids (see appendix). Their methods use the inverse map data structure to allow inexpensive determination of the curvilinear coordinates (i.e. interpolation coefficients) of a point. The specification of hole cutting surfaces is greatly simplified by using cartesian approximations of the body surfaces called hole maps. Dynamic hole expansion and shrinkage to allow minimization of overlap, or optimization of interpolation locations in the future, is also available. Wey modified an advancing-front technique for generating an unstructured mesh to efficiently and automatically assemble overset structured grids (ref. 6, also see appendix). His method considers the boundaries as a collection of vertices, edges, and facets, and is in principle quite flexible. Wey uses an implementation of the "enlarged orientation theorem" to efficiently solve the important problem of whether a point lies in the interior, exterior, or on the boundary of a prescribed surface or front. Other's have also attempted varying degrees of increased automation and decreased user input, including the author.

Maple and Belk have developed an integrated flow solver and overset mesh system assembler called the Beggar code (ref. 9, also see appendix). Blocked, patched, and overset grids are used by the Beggar code. A hierarchical arrangement is used to improve efficiency, with collections of blocked and patched grids grouped together in a superblock. Superblocks are allowed to overlap with other superblocks to form the overset grid system. The idealistic goal for this computer program was the ability to accept input solely consisting of:

1. Good grids.
2. Physical boundary conditions other than farfield, e.g. tangent flow, no slip wall, or mass flow at an inlet face, etc.
3. Flow solver and six degree of freedom integrator parameters, e.g. CFL number or time step size, number of subiterations, etc.
4. Specification of required output.

The code should then assemble the grids, determine the physical and numerical boundary conditions on all unspecified grid surfaces, and produce the solution. We have had fair success in meeting these goals, with the primary difficulty being in the definition of good grids.

The data structure used in the Beggar code to facilitate determination of holes and interpolation coefficients is a variant of the polygonal mapping (PM) tree (ref. 11, 12). Stencil jumping (Newton iteration) is used to determine precise interpolation coefficients, but this process requires good starting guesses, and can be very expensive to use to determine that a cartesian coordinate does not lie within a grid. The desired benefit from the Beggar data structure is to unambiguously provide a list of grids containing a point, and then to give a stencil jumping starting guesses for these grids guaranteed to converge. The PM tree satisfies these requirements in nearly all cases. Basically, an octree data structure provides gross subdivision of the solution domain into smaller regions that have the property of being entirely within a grid, entirely outside a grid, or on a grid boundary. Those regions that are entirely within a grid satisfy the requirement that a single start point is guaranteed to result in successful stencil jumping within the region. Those regions that are on the boundary contain a small binary space partitioning (BSP) tree that provides accurate in/out determination for all points in the region, as well as appropriate stencil jump starting points depending on the grid in which the point falls.

### Complexity

As discussed in the previous section, many new methods are being developed to automate the process of overset mesh assembly. This results in significantly less workload on the end user of these techniques, but this is not without some cost. The codes themselves tend to become more complex, and less inviting for the engineer or even the numerical algorithm specialist to delve into. The advanced algorithms having to do with overset mesh assembly are often computer science algorithms, not numerical algorithms. Overture++ is a C++ class library designed by Reider and Quinlan (see appendix) to lessen the difficulty of writing partial differential equation solvers using overset grids. This class library will offer utility routines to perform tasks that are generic to all overlapping grid operations, including memory management. Pao (see appendix) gave results for a compressible low speed flow algorithm implemented using Overture++.

## SUMMARY AND DISCUSSION

Reviewing the material that has been presented here, it is evident that the push for more accuracy, efficiency, and usability of overset methods is slowly, but fundamentally, changing the overset community's approach. Formerly pure overset methods are allowing combination with blocked, patched, and cartesian structured grids. Even more revolutionary for this community is the use of unstructured and prismatic grids in some regions. Communication between all these disparate grid types is usually done in one of two ways: 1. the

traditional hole cutting, overlap, and interpolation (Schwartz alternating procedure), or 2. Elimination of overlap and use of unstructured grid mortar across the gap. The other fundamental change is in embracing more complex data structures and advanced algorithms. This includes use of unstructured grids, octrees, BSP trees, inverse maps, alternate digital trees, and C++ class libraries.

The original impetus for the use of overset grids was the desire to model complex geometry using body fitted curvilinear coordinates. The rationale for this was accuracy, efficiency, and code reuse. Accuracy -- because body fitted coordinates allowed accurate boundary condition imposition, provided for viscous layers, etc. Efficiency -- since the computational rectangle associated with each structured component grid allowed vectorization. Code reuse -- because each grid was just another computational rectangle to the solver. Another historical motivation was that relative rigid motion of bodies could be modeled with no regridding required.

The environment is of course different now than fifteen years ago. There are still strong arguments for the use of overset methods, at least for the generalized overset methods that have come into use recently. One argument that has stayed the same is the usefulness of overset methods in modeling moving bodies without regridding. Overset methods still offer advantages in accuracy and efficiency. Accuracy and efficiency can no longer be taken as strictly synonymous with structured grids and vectorization, however. Instead the argument is that overset methods allow use of the most appropriate mesh type and physics model in each domain of the problem. Adaptive refinement via additional meshes is possible. Also, parallelization by domain decomposition is natural and does not require a homogeneous processor environment. Code reuse is still a real advantage of overset methods. Here, too, the emphasis has changed. No longer does this imply that a single solver package should be reused on all the different grids. Instead, when putting disparate mesh types and physics models together, the solvers developed for structured, unstructured, cartesian meshes, and so on can be linked together with relatively minor modifications to operate on a generalized overset grid system.

## CONCLUSION

Overset methods are changing to meet the demands of current problems and to take advantage of available technology. The overset techniques and computer codes can usually incorporate different mesh types and physics models without major restructuring of either the overset code or the stand alone version of the new solver or model. Because of this, the overset codes are ideally suited to growing into the "general purpose" CFD codes of the future.

# APPENDIX

The following are the presentations given at the 2nd Overset Composite Grid and Solution Technology Symposium, held at Fort Walton Beach, FL on October 25-28, 1994. Copies of particular abstracts and presentation material are available for a nominal fee from the symposium General Chair, Dr. Lawrence Lijewski. He can be reached at phone (904)882-3124, ext 3376, e-mail lijewski@eglin.af.mil, or

WL/MNMW
101 W. Eglin Blvd. Suite 346A
Eglin AFB FL 32542

## Presentations

Introductory Remarks - J. Benek

Research Topics in Computational Methods for Unsteady Multiple-Body Aerodynamics - R. L. Meakin, Overset Methods Inc.

Issues and Advances in Overlapping Grid Generation - G.S. Chesshire and W.D. Henshaw, Los Alamos National Laboratory

A New Approach to Domain Decomposition: The Beggar Code - D. Belk and R. Maple, Wright Laboratory

Overture++: A C++ Class Library for Overlapping Grid Solvers -M. Reider, Los Alamos National Laboratory

Two New Chimera Methods: Application to 3D Store Separation - J.P. Gillyboeuf, Aerospatiale-Missiles; P. Mansuy, Matra Defense; and S. Pavsic, ONERA; France

A Method for Computing the 3D Flow Around a Ship using Composite Overlapping Grids - J. F. Malmliden, Royal Institute of Technology, Stockholm, Sweden

Chimera Grid Application for Fighter Configurations - M. Mani, McDonnell-Douglas

Verification of a Transonic Euler Solution of an F-16 Aircraft with a Generic Finned Pressure-Instrumented Store Using Chimera Grid Scheme - W.C. Riner, B.A. Jolly, N.C. Prewitt, Sverdrup Technology Inc.; and J.M. Brock, Jr., Test Wing, Eglin AFB, Florida

On Automating Domain Connectivity for Overset Grids - I.T. Chiu and R.L. Meakin, Overset Methods Inc

Development of an Automatic Mesh Interface Generator for Overlapped Structured Grids - T. C. Wey, NASA Johnson Space Center

Tutorial: PEGSUS - N.E. Suhs

Computational Fluid Dynamics for Multiple Projectile Configurations - J. Sahu and C.J. Nietubicz, Army Research Laboratory

Numerical Simulation of a Paper Coating Flow - F. Olsson, Royal Institute of Technology, Stockholm, Sweden

Computational Methodology for Time-Accurate Multiple Body Motion -R.D. Thoms and J.K. Jordan, Calspan Corp

The Prediction of Unsteady HEDI Shroud Separation Event - J.P. Narain, Lockheed Missiles and Space Co.

Tutorial: BEGGAR - D. Belk

Conservation and Linear System Issues on Overset Composite Grids - J.S. Saltzman, Los Alamos National Laboratory

An Advance in Overset Grid Schemes: From Chimera to DRAGON Grids, - K.H. Kao and Meng-Sing Liou, NASA Lewis Research Center

Conservative Chimera for 3D Euler Equations on Structured/Structured, Structured/Unstructured Grids, - Z.J. Wang, CFD Research Corp

Tutorial: DCF3D, R.L. Meakin

Comparisons of Overlapping Grid Communications with Beggar and Pegsus - N.C. Prewitt, Sverdrup Technology, Inc

Navier-Stokes Analysis for Propulsion-Airframe Integration by Using OVERFLOW/Chimera Overset Grid Approach - L.M. Gea, McDonnell- Douglas Corp.

Some Experiences with the NPARC Overset Grid Capability - H.J. Thornburg, B.K. Soni, M.H. Shih, B.K. Kishore, Mississippi State University

General Approach to Calculating Forces and Moments on Overset Grid Configurations - W.E. Dietz, Calspan Corp

Recent Developments in Grid Generation and Force Integration Technology for Overset Grids - W.M. Chan, NASA Ames Research Center

Adaptive Composite Overlapping Grids for Hyperbolic Conservation Laws - K.D. Brislawn, D.L. Brown, G.S. Chesshire and J.S. Saltzman, Los Alamos National Laboratory

Adaptive High-Order Godunov Projection Methods for the Incompressible Navier-Stokes Equations on Overlapping Grids - D.L. Brown and W.J. Rider, Los Alamos National Laboratory

A Structured/Unstructured Overset Grid Flow Solver for Helicopter Rotor Flows with Adaption - E.N. Duque, Army ATCOM, NASA Ames Research Center

Parallel Adaptive Mesh Refinement for Overlapping Grids - D. Quinlan, Los Alamos National Laboratory

A Navier-Stokes Chimera Code on the Connection Machine CM-5: Design and Performance - D.C. Jespersen and C. Levit, NASA Ames Research Center

Progress Report on High-Performance High-Resolution Simulations of Coastal and Basin-Scale Ocean Circulation - D.W. Barnette, J.M. Swisshelm, R. Tuminaro and C.C. Ober, Sandia National Laboratory

Tutorial: CMPGRD - G. S. Chesshire

CGINS: A Solver for the Incompressible Navier-Stokes Equations on Overlapping Grids - W.D. Henshaw, Los Alamos National Laboratory

An Algorithm for All Speed Flows - K. Pao, Los Alamos National Laboratory

Analysis of the Space Shuttle Ascent Aerodynamic Environment - R.J. Gomez and F.W. Martin,Jr., NASA Johnson Space Center

GBU-28 Pressure Port Analysis - B.A. Jolly, Sverdrup Technology Inc.; J.M. Brock,Jr., Test Wing, Eglin AFB; and L. Coleman, Tybrin Corp.

Analysis of the Re-Designed Space Shuttle APU Control Value - C.H. Campbell and T.C. Wey, NASA Johnson Space Center

## REFERENCES

1. Meakin, R.: On the Spatial and Temporal Accuracy of Overset Grid Methods for Moving Body Problems. AIAA Paper 94-1925, June 1994.

2. Kao, K. H. and Liou, M. S.: Direct Replacement of Arbitrary Grid-Overlapping by Nonstructured Grid. AIAA Paper 95-0346, January 1995.

3. Johnson, R. A. and Belk, D. M.: A Multigrid Approach to Embedded-Grid Solvers. AIAA Paper 93-0769, January 1993.

4. Rogers, S. E. and Pulliam, T. H.: Accuracy Enhancements For Overset Grids Using a Defect Correction Approach. AIAA Paper 94-0523, January 1994.

5. Chesshire, G., and Henshaw, W. D.: Composite Overlapping Meshes for the Solution of Partial Differential Equations. JCP, vol. 90, pp. 1-64, 1990.

6. Wey, T. C.: Development of a Mesh Interface Generator for Overlapped Structured Grids. AIAA Paper 94-1924, October 1994.

7. Lijewski, L. E. and Suhs, N.: Chimera-Eagle Store Separation. AIAA Paper 92-4569, August 1992.

8. Chan, W. M. and Buning, P. G.: A Hyperbolic Surface Grid Generation Scheme and its Applications. AIAA Paper 94-2208, June 1994.

9. Maple, R. C. and Belk, D. M.: Automatied Set Up of Blocked, Patched, and Embedded Grids in the Beggar Flow Solver. Numerical Grid Generation in Computational Fluid Dynamics and Related Fields, ed. N. P. Weatherill et al., 1994, Pine ridge Press, pp. 305-314.

10.Chan, W. M., Chiu, I. T. and Buning, P. G.: User's Manual for the HYPGEN Hyperbolic Grid Generator and the HGUI Graphical User Interface. NASA TM 108791, October 1993.

11.Samet, H.: Applications of Spatial Data Structures: Computer Graphics, Image Processing, and Other Areas, Addison-Wesley, 1989.

12.Samet, H.: Design and Analysis of Spatial Data Structures. Addison-Wesley, 1990.

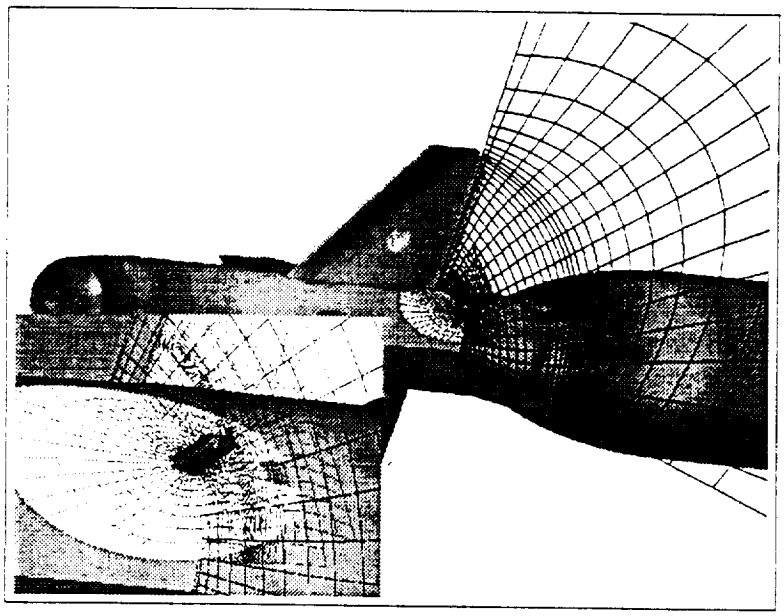


Figure 1: Use of an overset grid to add an appendage to an existing configuration.

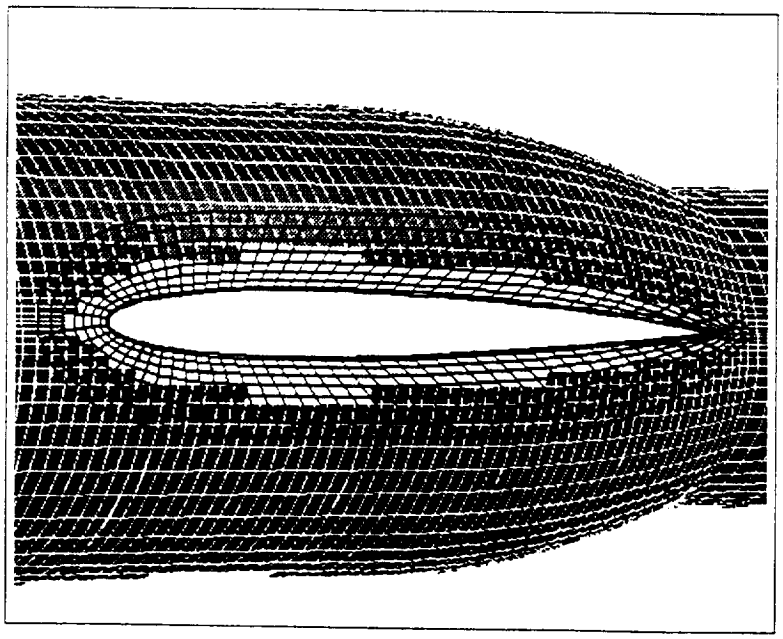


Figure 2: Example of problem area for force integration on intersecting surfaces.

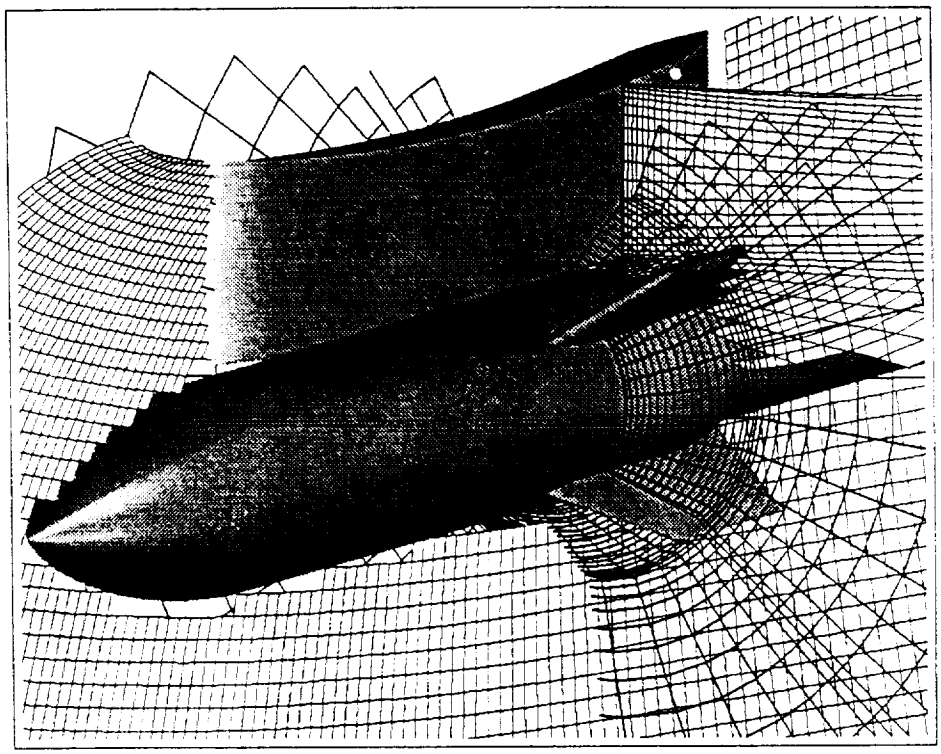


Figure 3: Beggar grid with blocked grid for finned store.

# CARTESIAN GRID TECHNOLOGY

N95-28739

19951223 18

# A CARTESIAN, CELL-BASED APPROACH FOR ADAPTIVELY-REFINED SOLUTIONS OF THE EULER AND NAVIER-STOKES EQUATIONS

**William J. Coirier***
**NASA Lewis Research Center, Cleveland, Ohio**

**Kenneth G. Powell****
**The University of Michigan, Ann Arbor, Michigan**

## ABSTRACT

A Cartesian, cell-based approach for adaptively-refined solutions of the Euler and Navier-Stokes equations in two dimensions is developed and tested. Grids about geometrically complicated bodies are generated automatically, by recursive subdivision of a single Cartesian cell encompassing the entire flow domain. Where the resulting cells intersect bodies, N-sided "cut" cells are created using polygon-clipping algorithms. The grid is stored in a binary-tree data structure which provides a natural means of obtaining cell-to-cell connectivity and of carrying out solution-adaptive mesh refinement. The Euler and Navier-Stokes equations are solved on the resulting grids using a finite-volume formulation. The convective terms are upwinded: A gradient-limited, linear reconstruction of the primitive variables is performed, providing input states to an approximate Riemann solver for computing the fluxes between neighboring cells. The more robust of a series of viscous flux functions is used to provide the viscous fluxes at the cell interfaces. Adaptively-refined solutions of the Navier-Stokes equations using the Cartesian, cell-based approach are obtained and compared to theory, experiment and other accepted computational results for a series of low and moderate Reynolds number flows.

## I. INTRODUCTION

For complicated geometries, unstructured grids can be easier to generate than structured grids, which is directly responsible for their increasing popularity. Traditionally unstructured grids, with volume grids comprised of triangles in two-dimensions and tetrahedra in three-dimensions, are typically generated by first discretizing the bounding surfaces and then filling the volume grid by, say, an advancing front method, or, by triangulating a cloud of points. Both approaches are based upon a user specified surface discretization which is closely coupled to the volume grid generation by requiring the specified faces on the boundary surfaces to be faces of cells in the volume grid. In the approach considered here, the volume grid and surface description are not strongly coupled in this manner. The computational boundaries are described functionally, and are "cut" out of the automatically generated, Cartesian-cell based volume grid, yielding N-sided cells near the boundaries. This yields a computational grid comprised mostly of square, unit aspect ratio (Cartesian) conservation volumes with polygonal boundary cells. The ability to automatically create the volume grid and the cut, boundary cells gives the Cartesian-cell approach its utility, but adds some complexity to the numerics and resulting computer code. Since the cell geometry and hence the cell-to-cell connectivity for all cells is *not* known apriori, a unique data structure is needed to describe the conservation volumes. Indeed, it is this complication that sets the Cartesian-cell based approach apart from most of the traditional unstructured grid approaches that are becoming more and more prevalent today.

The Cartesian, cell-based approach has been used for solving a variety of problems, and variants of the approach have proven to be quite useful for computing both unsteady [6,7,26,27,5,10,25] and steady

* coirier@lerc.nasa.gov
** powell@engin.umich.edu



PRECEDING PAGE BLANK NOT FILMED

[11,8,9,16,15,19,22] adaptively-refined solutions to the Euler equations and for solutions to the transonic full potential equation [33]. Recently, the usefulness of the automated gridding and adaptive mesh refinement capabilities of the solver have been demonstrated for adaptively-refined solutions of the Euler equations about large aircraft configurations [23]. The automated mesh generation combined with the adaptive-mesh refinement capability of this approach offers the potential of obtaining accurate solutions with minimal user intervention. The work shown here investigates the extension of this capability to solving the compressible Navier-Stokes equations in an upwinded, finite-volume framework.

## II. GRID GENERATION PROCEDURE

The grid is generated by the recursive subdivision of a single cell, and during the creation of the grid, the hierarchical relation between newly created cells and their parents are stored in a binary tree data structure. The cut cells, which are the background Cartesian cells *cut* into polygons, are created automatically using many concepts borrowed from computer graphics applications. Since they are hierarchically related to their Cartesian parents, they are also stored in the tree. This procedure of cell cutting is a subject unto itself and its robustness is absolutely crucial for the utility of this approach. The cell cutting methodology used here is based upon a polygon clipping algorithm [31], where a subject polygon (the body of interest) is "clipped" against a convex clipping polygon, which for this case is the Cartesian cell. The subject polygon can be formulated to include arbitrary functional descriptions of the bounding faces (i.e.-geometric description), which is used here. This particular clipping operation yields the logical and operation between the clipping polygon and the subject polygon. This operation will yield the correct cells when the subject polygon describes the outer boundary of a flow domain, but needs to be modified slightly when the subject polygon describes an inner boundary. When this is the case, the region needed is recovered from the clipping operation using a list directed vertex insertion procedure. Details behind the grid generation are presented in more detail in [12]. Once a suitable geometric description of the computational boundaries is made, the grid generation is automatic, and since the procedure is recursive and tree based, it is also efficient. An example of the grid generation is shown in Figure 1, where a coarse, base grid in a flow passage representative of the cooling passage within a turbine blade is shown. The geometry corresponds to that in [29], and is a projection onto the x-y plane of the geometry located along a curved surface about the turbine blade mean chord line. The grid contains 2640 cells and was generated in 218 seconds on an IBM RS6000 Model 560 workstation. The geometry definition is made using a linearly represented continuous outer boundary and 14 cooling fins.

The use of this particular grid and data structure easily allows adaptive mesh refinement, and away from bodies yields smooth grids. The binary tree data structure provides a logical means of finding cell-to-cell connectivity by logic based tree traversals and allows a straightforward means to perform mesh refinement and coarsening via tree branch growth and pruning. There are many niceties afforded by this data structure and grid setup. The grid hierarchy is amenable to multi-grid [17] and provides a natural means of domain decomposition that appears well suited for coarse grain parallel computations.

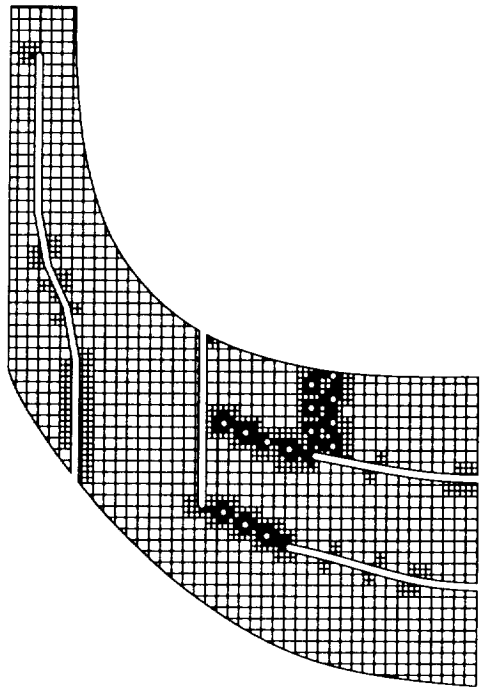


Figure 1 Representative
Turbine Cooling Passage Grid

## III. SOLUTION OF THE EULER EQUATIONS USING A CARTESIAN,CELL-BASED APPROACH

The Euler equations are solved using a cell-centered, finite-volume, upwinding approach. A limited, linear reconstruction of cell-averaged data is used to provide input to a numerical flux function, yielding the flux through cell-to-cell interfaces. The numerical fluxes are computed in an upwind fashion using an appropriate approximate Riemann solver. These fluxes are then used to perform a flux balance upon the conservation volume, which is then used to advance the conserved variables in time. The procedure follows standard practice for a finite-volume scheme. The solution procedure can be broken up into 3 stages; reconstruction, flux construction and then evolution to steady-state. A more detailed description of the procedures is shown in [12].

The variation of the cell primitive variables in each cell is reconstructed using a linear reconstruction procedure, in the spirit of MUSCL interpolation, based on the Minimum-Energy reconstruction presented by Barth [3]. The Minimum-Energy reconstruction minimizes the Frobenius norm of the differences between the cell averages of the reconstructing polynomial and the cell averages of the support set. This reconstruction procedure is K-exact, in the sense that if a linear function is cell averaged upon the mesh, the reconstructed polynomial returns the same, exact, linear function. The Minimum-Energy reconstruction procedure provides the framework of extending the order of reconstruction, but only a linear reconstruction is considered here.

This reconstruction of cell averaged data does not preclude the introduction of new extrema: There is no means to ensure that the reconstructed solution is bounded by the data used to perform the reconstruction. To enforce this, the reconstruction is limited by evaluating the cell averaged data of the support cells used, reducing the reconstructed gradient to achieve monotonicity of the data. This will in turn guarantee mono-

tonicity of the solution if the numerical flux function is a positivity preserving function (which for an upwind scheme is sufficiently implied by positivity of the dissipation matrix), and provided that a proper choice is made for the time step. The concept of restricting the local solution to be bounded by its immediate neighbors is based upon a discrete interpretation of a local maximum principle, and has been used to evaluate the stencils obtained for a model equation of the viscous terms of the Navier-Stokes equations, in [12]. The limiting procedure implemented here is based upon the procedure shown by Barth and Jespersen [4]. To ensure monotonicity of the reconstruction at cell interfaces, the solution is required to be bounded by the data used to perform the reconstruction. A single limiter, $\Phi = min(\phi_j)$ is found for all the primitive variables and applied to the reconstructions.

The inviscid numerical fluxes may be computed using a variety of approximate Riemann solvers: It is a simple matter to supply a different numerical flux function by replacing the approximate Riemann solver in the flow solver. For the work shown here, most of the computations have been performed using the AUSM scheme of Liou and Steffen [21]. This novel flux function combines the efficiency of flux vector splitting with the accuracy of flux difference splitting. The derivation and use of this flux function is available in [21].

For simplicity, the semi-discrete form of the equations are advanced in time using a multi-stage scheme. A spatially varying time step is used, and is indeed quite necessary, since there is typically a many order variation in cell size across the mesh due to cell refinement and cutting. A generic multi-stage scheme is used to advance the solution from the n-th to the (n+1)-th time level.

## III.a SOLUTION ADAPTIVE MESH REFINEMENT

The Cartesian, cell-based approach gains its strength primarily from two features; the ability to compute flows about complicated geometries where the initial grid is obtained automatically, and by the inherent ease in which adaptive mesh refinement can be performed. Adaptive mesh refinement is an attempt to improve the quality of a solution by adding cells locally where an increased resolution is desired, and by possibly removing cells where the current resolution is unnecessarily too high. This feature, coupled with the automated means of mesh generation, attempts to yield grid converged solutions about geometrically complicated domains with minimal user intervention.

Each level of adaptive mesh refinement is comprised of two stages. In the first stage, refinement criteria are constructed for all cells on the mesh, and then in the second stage, cells are tagged for refinement or coarsening based on this criteria. After the mesh is enriched, a new calculation is made, converging the solution to a steady, and hopefully more accurate solution. This process of refining the grid and converging the solution on the new grid is repeated in an automated fashion, a set number of times, until a given level of refinement is achieved.

The refinement criteria and grading procedure used here are based upon that presented in [14,15]. Unless otherwise noted, no changes to the form of the refinement criteria or the selection levels are made. The procedure of refinement and coarsening of the cells is based upon a statistical description of the cell size weighted velocity divergence and curl. The local velocity divergence is used to detect compressive phenomena, while the velocity curl is used to detect shear. Each of these is weighted by the local cell size so that smaller cells contribute less to the overall weighting, as suggested in [32].

## III.b VALIDATION OF THE EULER SOLVER

The Cartesian, cell-based approach has been demonstrated extensively for a wide variety of inviscid flows [11,8,9,16,15,19,22,23] and has been assessed for accuracy [13]. In [13] the approach was shown to compute transonic flows with a global second-order accuracy and a local accuracy between first- and second-

order. The adaptively refined solutions about a transonic single-component airfoil and a sub-critical, multi-component airfoil are shown here, demonstrating the adaptive-mesh and mesh generation capability.

## III.c AGARD CASE 06: TRANSONIC FLOW PAST A RAE 2822 AIRFOIL

This case corresponds to the same geometry and free-stream conditions as Test Case 06 in the collection of inviscid flow test cases[1]. Adaptive mesh refinement is performed for four levels beyond the base grid level. Figure 2 shows the final, adapted grid and Figure 3 shows the Mach number contours at the final refinement level.The solution compares well with the computed results, tabulated in [1]. Figure 4 shows the computed surface Mach number and the computed results from [1].

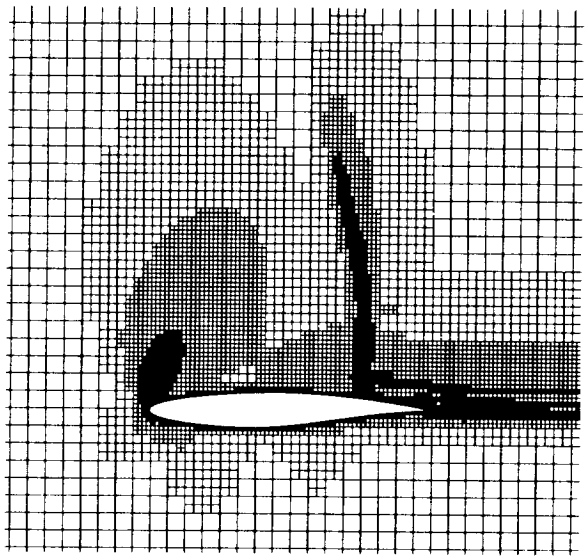


Figure 2 RAE 2822 Level 4 Adapted Grid

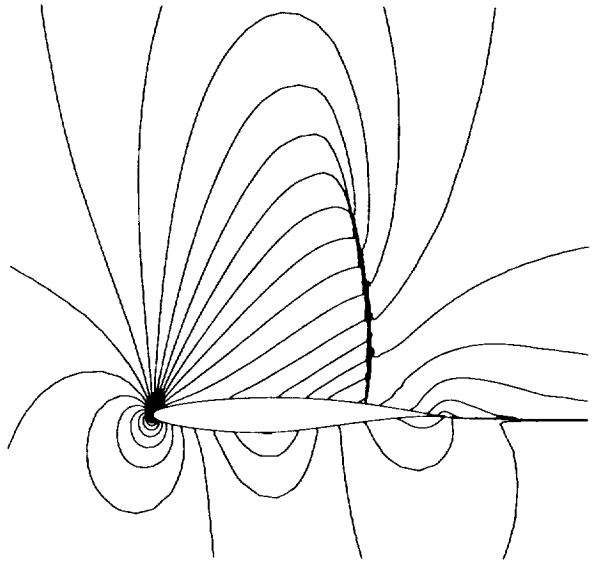


Figure 3 RAE 2822 Level 4 Adapted Mach Number Contours

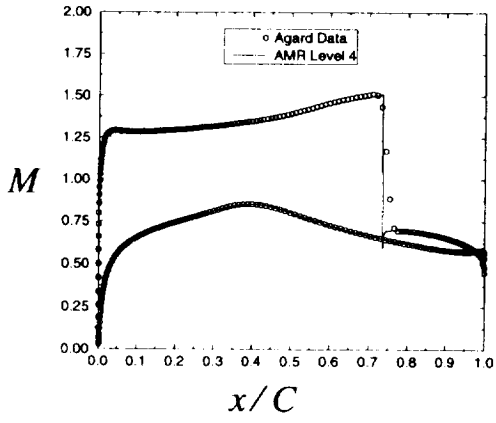



Figure 4 Comparison to Computed Surface Mach Numbers from AGARD 211

## III.d SUDDHOO-HALL MULTI-ELEMENT AIRFOIL

This four-element airfoil has been included in a series of test cases at the 1994, ICASE/NASA LaRC Workshop on Adaptive Grid Methods. The geometry corresponds to that obtained by a conformal mapping technique [30], which has also yielded surface pressure data. The airfoil geometry has been curve fit using a cubic spline and made available on Mosaic (http://www.icase.edu/workshops/adapt) by the workshop organizers. The free-stream conditions corresponding to this case are $M_\infty = 0.2$ and $\alpha = 0°$. The mesh is generated automatically and adaptive-mesh refinement performed for 3 levels beyond the base grid. The computed solution on the final grid is compared to the potential flow solution in Figure 7 while the final grid and pressure contours are shown in Figure 5 and Figure 6.

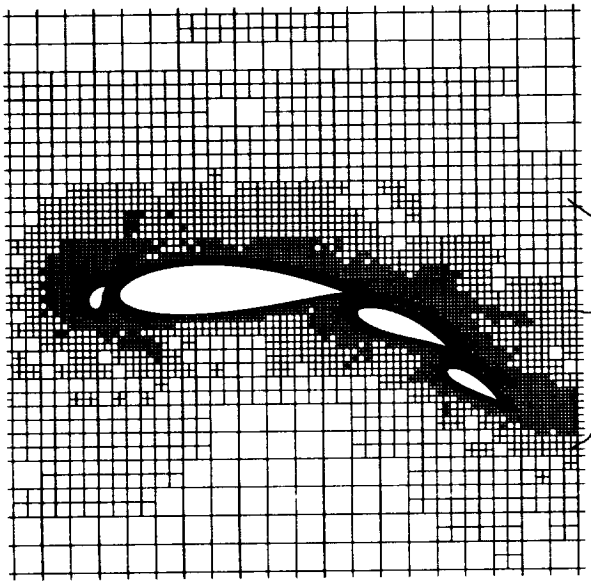


Figure 5 Level 3 Adapted Grid

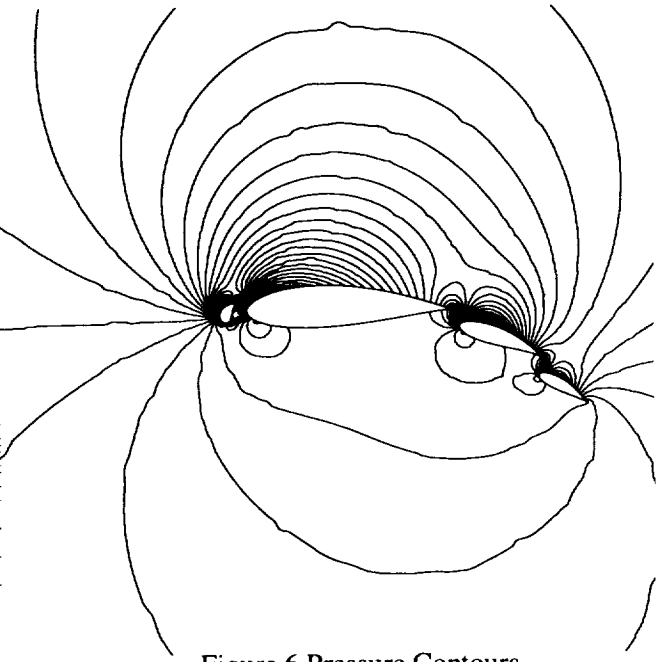


Figure 6 Pressure Contours

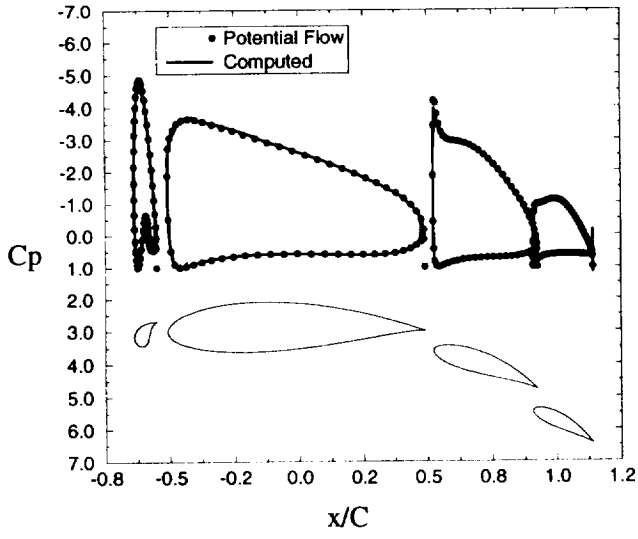



Figure 7 Computed Pressure Coefficients

# IV. VISCOUS FLUX FORMULATION

The essence of the cell-centered viscous flux formulation is the reconstruction of the gradients of the velocity and temperature at the cell interfaces from the cell-averaged data of the cells around it. Once this has been completed, a numerical quadrature is performed over the cell faces, yielding the desired viscous fluxes through the interfaces. For cell-centered schemes, there are prevalently two separate classes of the viscous gradient reconstructions, which have been analyzed in [12]. The first and most widely used class is based upon an application of the divergence theorem to a co-volume surrounding the face where the flux is desired. These types of reconstructions are classified as Green-Gauss type reconstructions. Four of these types have been analyzed in [12], and are delineated amongst themselves by the reconstruction co-volumes and the procedures used to obtain the data at the vertices of the co-volumes. A different class of gradient reconstructions based upon expanding a polynomial about the face midpoint and then differentiating the polynomial to obtain the gradients has been proposed by Mitchell [24]. Linear and quadratic reconstructing polynomials are found using a Lagrangian type of interpolation, and are also examined in detail in [12].

There, the six schemes were analyzed for quality and positivity by local Taylor-series expansions of the stencils created for Laplace's equation on grids representative of the Cartesian approach. It was shown the importance of the K-exactness of the interface gradient reconstructions. The analysis showed that for arbitrarily distorted grids, the only means to obtain a conservative, first-order accurate discretization requires that the gradients be found from quadratically-preserving functions. Importantly, it was shown that if the gradient reconstruction procedure is not at least linearity-preserving, stencils can be obtained that preclude grid convergence, and will actually have an error that *increases* with mesh refinement. This type of truncation error is termed here as being mesh divergent. Very importantly, robustness of all of the schemes upon the distorted meshes caused by mesh refinement was gauged using a positivity analysis. Non-positivity of a reconstruction scheme can inhibit convergence and violates the discrete interpretation of the continuous maximum principle entertained by elliptic solutions.

To summarize the analytical results presented in [12], of the four Green-Gauss type reconstructions analyzed, three were shown to be either mesh divergent, produced either extremely non-positive stencils or gave solutions that on certain topologies yielded stencils that were completely or partially decoupled from the neighboring cells. The fourth reconstruction type is commonly known as a diamond-path reconstruction using a linearity-preserving weighting. This reconstruction is named so because it uses a diamond-shaped polygon to perform the gradient reconstruction. The four vertices of this polygon are formed by the two cell centroids that share the face to be reconstructed about, and the two vertices that are at the face endpoints. The data at the centroids is known exactly, but the data at the vertices must be arrived at by some interpolating procedure. In [20] a linearity-preserving weighting is derived which finds data at a vertex using the local cells about it. Since the Green-Gauss reconstruction procedure can only preserve linear functions, and can only do this if the data supplied to it is linearity-preserving, this weighting procedure guarantees the reconstruction of a gradient that is linearly K-exact. A means of obtaining higher-order preserving weightings was shown in [12], but for the linearity-preserving reconstruction scheme using the divergence integral, this higher-order weighting is unnecessary. Although the analysis showed that this reconstruction procedure was not positive on general meshes and could yield inconsistent stencils, it was seen to represent the better of the Green-Gauss class of reconstructions.

Of the polynomial reconstruction types, a Lagrangian type of interpolation procedure is used to reconstruct a polynomial based at the face midpoint, which is then differentiated to obtain the gradients. The reconstruction requires the selection of either three cells (for the linear reconstruction) or six cells (for the quadratic), which then necessitates the inversion of a Vandermonde-type matrix. In [24] it is suggested to base the selection of these support cells to provide a centroid of the resulting polygon to be closest to the face midpoint. In [12] it was shown that this criterion does not always yield the best stencil, and for the linear reconstruction scheme, yielded some extremely poor stencils on certain grid topologies. The analysis showed that the quadratic reconstruction was the only procedure that gave first-order accurate stencils for

Laplace's equation on arbitrarily distorted meshes. The analysis also indicated that this procedure could give the most non-positive stencils.

Since the analysis presented no clear choice as to the best procedure of the two classes of reconstructions, Green-Gauss or polynomial based, the two schemes were both used to compute a series of low to moderate Reynolds number, adaptively-refined solutions using the Cartesian approach. Although neither scheme guaranteed positive stencils, the diamond-path scheme could be viewed as representing the more positive, yet less accurate of the two. In [12], a discrete accuracy and positivity analysis on the grids showed that the inconsistency incurred by the diamond-path schemes is low while the computed results from both schemes were nearly identical. Globally this inconsistency incurred by the diamond-path scheme is small due in a large part by the regularity of the Cartesian grids. The quadratic polynomial based scheme guaranteed consistency, which was also shown computationally in the discrete accuracy analysis, but yielded the most non-positive stencils, which also made the quadratic scheme the least robust. For arbitrarily cut cells, neither scheme was as robust as would be desired, but the diamond-path scheme was shown to be the more positive. An important conclusion from this study is that the current viscous flux functions for cell centered schemes rely heavily upon grid smoothness and orthogonality to obtain accuracy and positivity. For the Cartesian grids, these properties do not hold at refinement boundaries, and are extremely violated near cut cells. This has negative implications for the smoothness of aerodynamic parameters that rely upon derivative quantities at walls, such as skin friction and heat transfer, and can have a detrimental effect upon convergence. Regardless of these comparatively negative findings, the approach can still prove to be useful, and can give accurate, automatically gridded and adaptively-refined solutions of the Navier-Stokes equations for low and moderate Reynolds number flows. The following computations illustrate adaptively refined solutions using the Cartesian-cell approach with the diamond path, linearity-preserving viscous flux function.

## IV.a LAMINAR DRIVEN CAVITY FLOW

The laminar flow inside a square, driven cavity is computed and compared to the computed results of Ghia, et. al. [18]. In [18], an incompressible formulation of the Navier-Stokes equations were solved using an implicit multi-grid method, where tabulated u- and v-velocity data is supplied along the lines through the geometric center of the cavity. To compare with these incompressible results, the Mach number used here is taken to be $M_{lid} = 0.1$. Two Reynolds numbers were computed and compared to the tabulated results.

### IVa.1 Re=100

A uniform base grid of 1024 cells (32 by 32) is generated, and three levels of adaptive mesh refinement beyond the base grid are obtained. Adaptive mesh refinement improves the solution slightly, but the initial solution is quite good. Figure 8 and Figure 9 show the computed u- and v-velocity profiles along vertical and horizontal lines through the geometric center of the cavity for the diamond path scheme. Figure 10 shows the final adapted grid and Figure 11 shows contours of u-velocity. Particle paths, showing the primary and secondary vortices is shown in Figure 12.

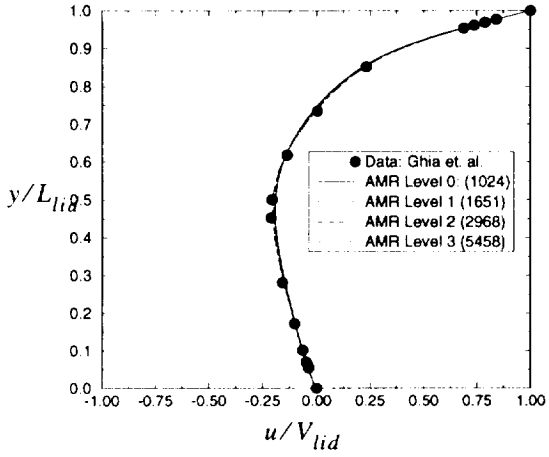



Figure 8 u-velocity Along Vertical Line Through Geometric Center

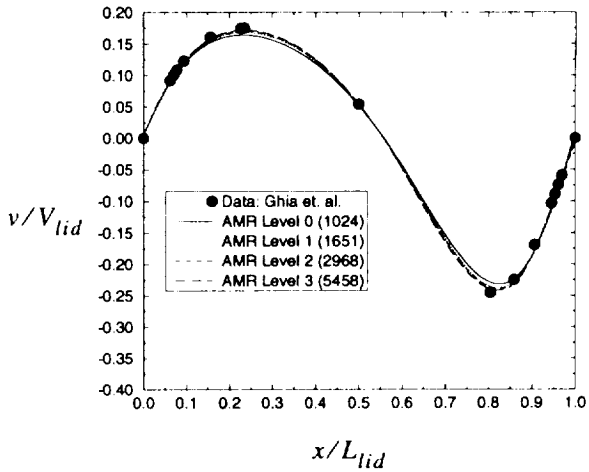



Figure 9 v-velocity Along Horizontal Line Through Geometric Center

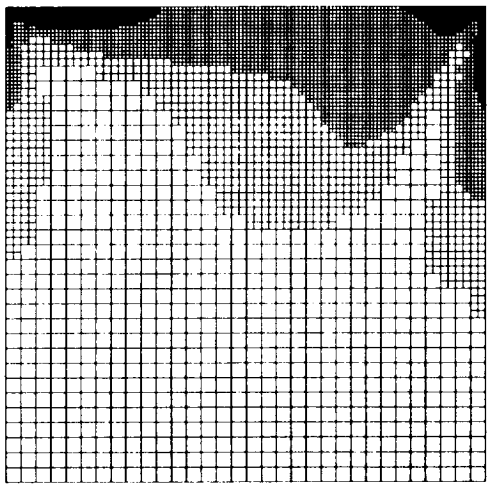


Figure 10 Refinement Level 3 Adapted Grid, Re=100 Case

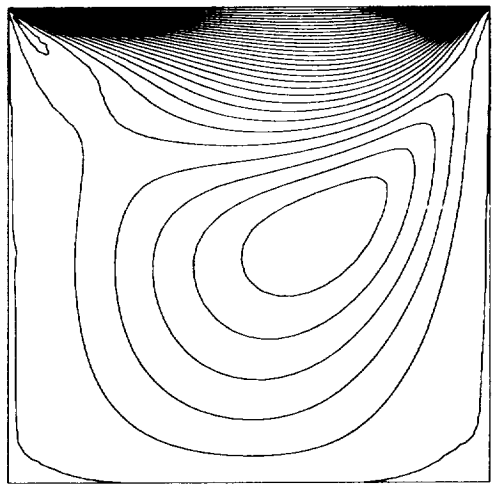


Figure 11 Refinement Level 3, u-velocity Contours

As seen in the adapted grid, the refinement strategy has added points near the lid and has resolved the grid near the upper corners, where there are singularities in the u-velocity. There are secondary vortices situated in the lower corners of the cavity: These vortices are not isolated by the refinement strategy, which indicates that a better strategy more suited for viscous flows might be needed. Overall, though, the solution is predicted well.

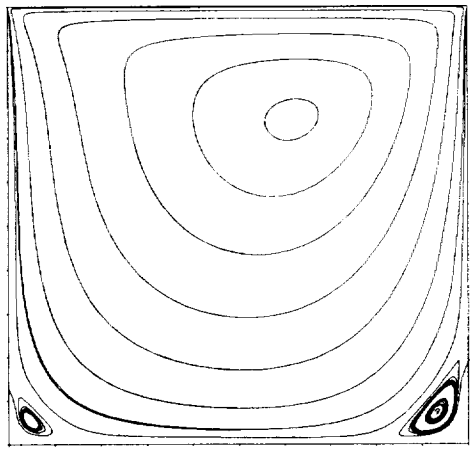


Figure 12 Particle Paths:

## IVa.2 Re=400

This case is similar to the previous, although the Reynolds number is now 400. A coarse base grid is generated, and three levels of adaptive mesh refinement are performed beyond the base level. A comparison to the computational data of Ghia[18] is shown for the u- and v-velocities on lines through the geometric center in Figure 13 to Figure 14.

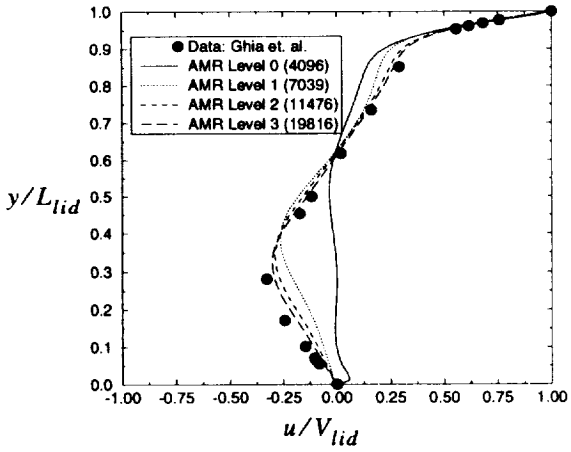



Figure 13 u-velocities on Vertical Line Through Geometric Center, Re=400

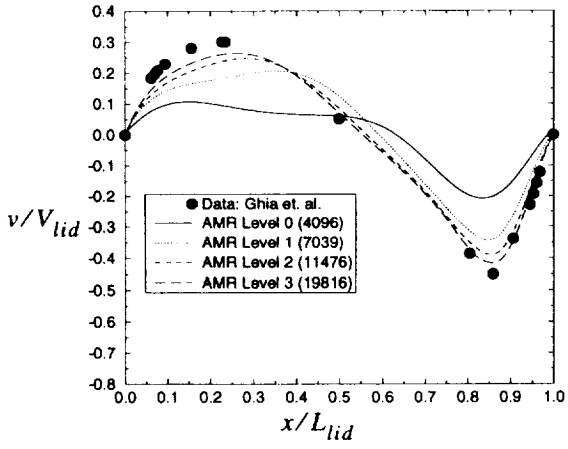



Figure 14 v-velocities on Horizontal Line Through Geometric Center, Re=400

Here, the solution on the coarse, base grid is poor, although the solution is improved through the adaptive mesh refinement. Figure 15, Figure 16 and Figure 17 show the adapted grid, u-velocity contours and particle paths in the cavity.

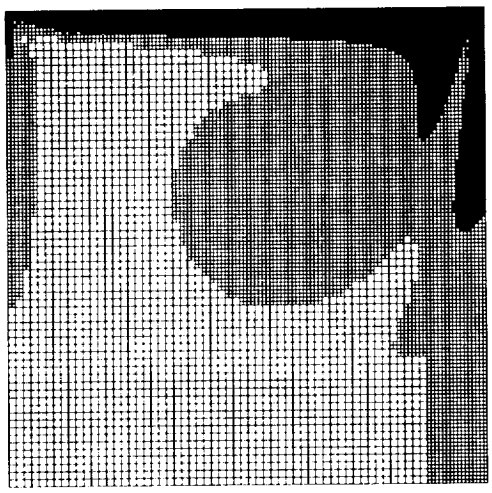


Figure 15 Final Adapted Grid, Re=400 Driven Cavity

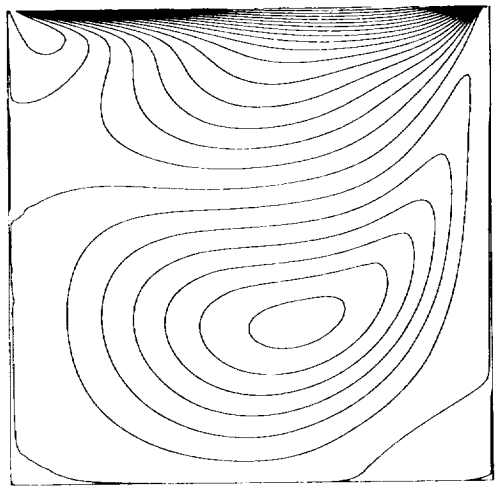


Figure 16 u-velocity Contours, Final Adapted

The solution obtained is good on the final grids, and the mesh refinement is shown to have improved the solution, even though the refinement strategy has not been derived with viscous flows in mind.

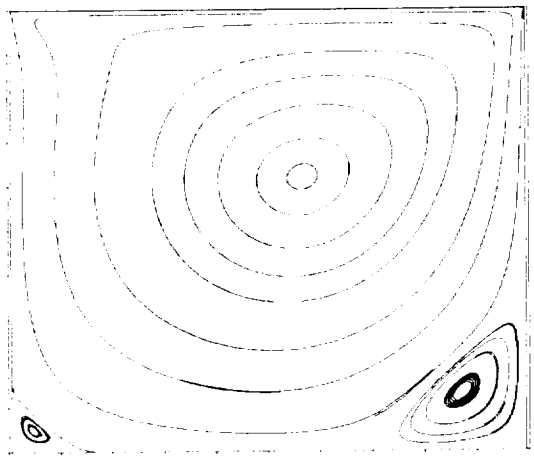


Figure 17 Particle Paths, Re=400 Driven Cavity

## IV.b LAMINAR FLOW OVER A BACKWARD FACING STEP

The laminar flow over a backwards facing step at two Reynolds numbers is used to validate the solver. The computed results are compared to the experimental data of [2] at the laminar Reynolds numbers. A parabolic velocity profile is specified at the inflow, and the exit pressure is specified. This ensures that the proper pressure gradient and mass flow is imposed on the flow.

## IVb.1 Re=100

Adaptive mesh refinement is made for three levels of refinement beyond the coarse, base grid. Figure 18 shows the grid and Figure 19 shows the effect of adaptive mesh refinement at a location corresponding to 2.55 step heights downstream of the step. Comparisons are made at other locations of the flow in [12]: The results compare equally well, and are not shown here. The agreement with the experimental data is good, and the adaptive mesh refinement improves the solution quality with each refinement.

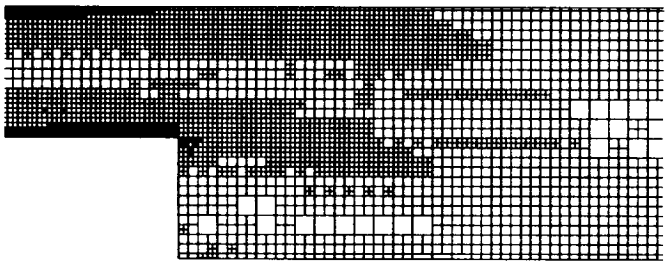


Figure 18 Portion of Adapted Grid at Refinement Level 3: Close-up Near Step

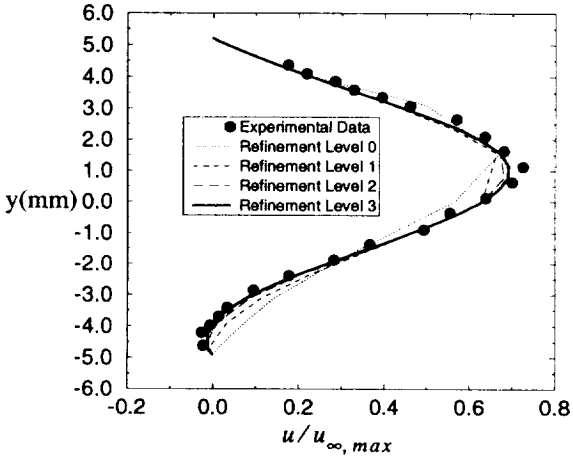



Figure 19 Comparison of Adapted Solutions at x/S=2.55. Re=100

## IVb.2 Re=389

This case is identical to the previous case, but the Reynolds number is Re=389. The computed velocity profiles through refinement at x/S=2.55 are shown in Figure 20. As before, the adaptive mesh refinement automatically improves the solution quality, and the final refinement level solution compares well with experiment.

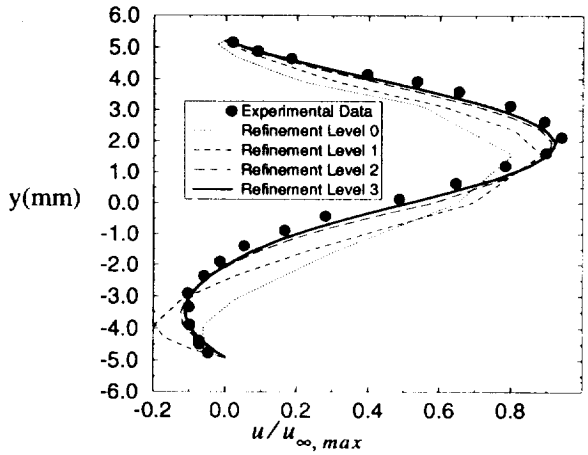



Figure 20 Comparison of Adapted Solutions at X/S=2.55. Re=389

## IV.c DEVELOPING LAMINAR FLOW OVER A FLAT PLATE

The developing, laminar flow over a flat plate which is aligned with the free-stream is computed and compared to the theoretical solution. A uniform flow is imposed ahead of the plate, and the flow is allowed to develop along the plate. Conditions are set so that the Reynolds number based on plate length is 10,000. The Mach number is taken to be $M_\infty = 0.2$ which eliminates the need for any compressibility transformation to compare to theory. When a poorly refined base grid is specified, the coarser solutions are improved successively by the adaptive mesh refinement. Figure 21 shows the close up of a base grid near the leading edge where the initial resolution of the grid is made according to an estimated normal velocity scale variation deduced from theory. Figure 22 and Figure 23 show the effect of adaptive mesh refinement upon the u- and v-velocity profiles while Figure 24 shows the skin friction through refinement.

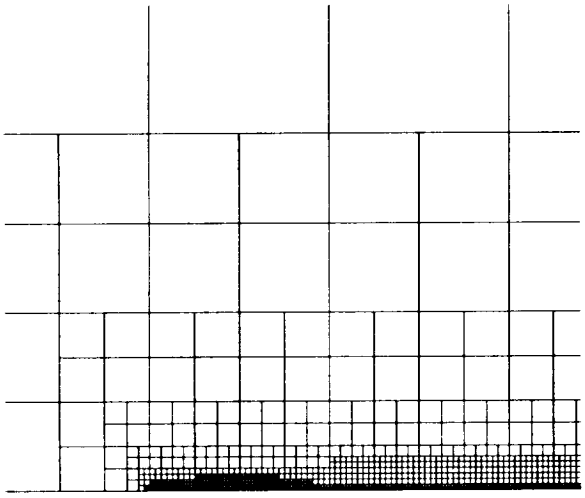


Figure 21 Close-up of Base Level Grid: Length Scale Smoothing: $\Delta u = 0.2$

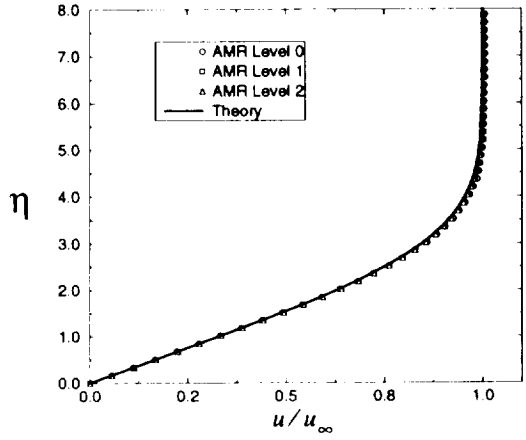



Figure 22 u-velocity Profiles: Effect of Adaptive Mesh Refinement at $Re_x = 8000$, Diamond Path Reconstruction.

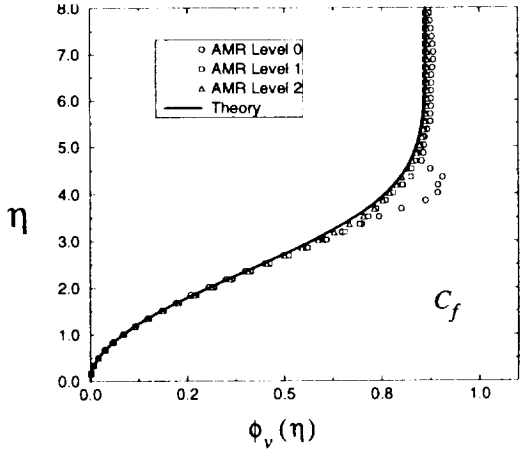



Figure 23 v-velocity Profiles: Effect of Adaptive Mesh Refinement at $Re_x = 8000$, Diamond Path

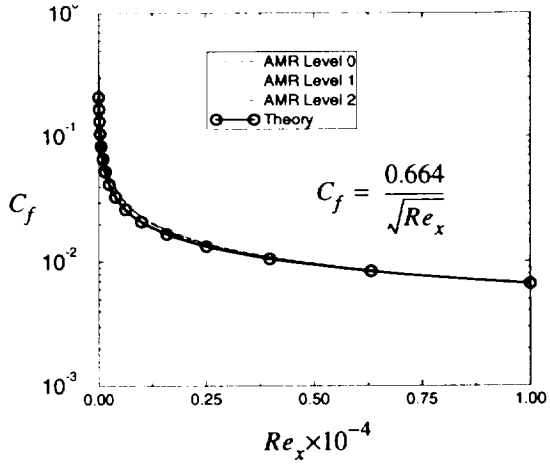



Figure 24 Skin Friction Through Adaptive Mesh Refinement, Diamond Path Scheme.

The smoothness of the skin friction in Figure 24 is due to the smoothness of the grid, since the root cell of the grid system has been located so that no cut cells are introduced along the surface of the flat plate. In [12], the plate is rotated 30° about the base axes and the identical flow is computed, bringing to light the effect of introducing cut cells along the plate boundary. The mean flow quantities are predicted well, but the skin friction is shown to be very oscillatory. This appears to be unavoidable with the current state of the art of the viscous flux functions. There is no flux function that guarantees both positivity and accuracy for cell centered schemes. Regardless of this finding, the mean flow quantities are predicted well, and the approach can be used to compute adaptively-refined solutions of the Navier-Stokes equations about complex geometries, as is indicated in the next case.

## IV.d LAMINAR FLOW THROUGH A BRANCHED DUCT WITH COOLING FINS

To demonstrate the approach for complex geometries, the flow in a stylized duct is computed. This duct geometry corresponds to an experiment conducted at NASA LeRC designed to simulate, in a simplified manner, the flow in the cooling passages of a turbine blade [28]. The calculations shown here in no way try to simulate the experiment: The experimental conditions correspond to a turbulent flow, while the calculations shown here are laminar. A schematic of the geometry and flow is shown in Figure 25.

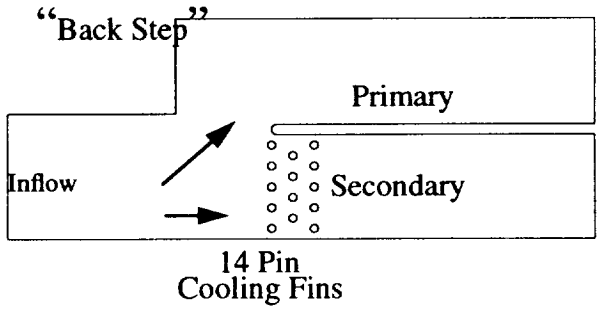



Figure 25 Schematic of Branched Duct Geometry

A fully developed profile is introduced at the inflow, and the flow is diverted into the primary passage by the blockage introduced by the pin fins in the secondary passage. Two different Reynolds numbers based on pin diameter and maximum velocity in the fully developed inflow profile were computed in [12]. The lower Reynolds number results are only shown here for brevity, where the Reynolds number based on maximum inflow velocity and pin fin diameter is Re=25. Only one level of adaptive-mesh refinement beyond the base grid level was obtained, due to positivity problems in the rear stagnation region of one of the pin fins. The final adapted grid and contours of total velocity are shown Figure 26 and Figure 27.

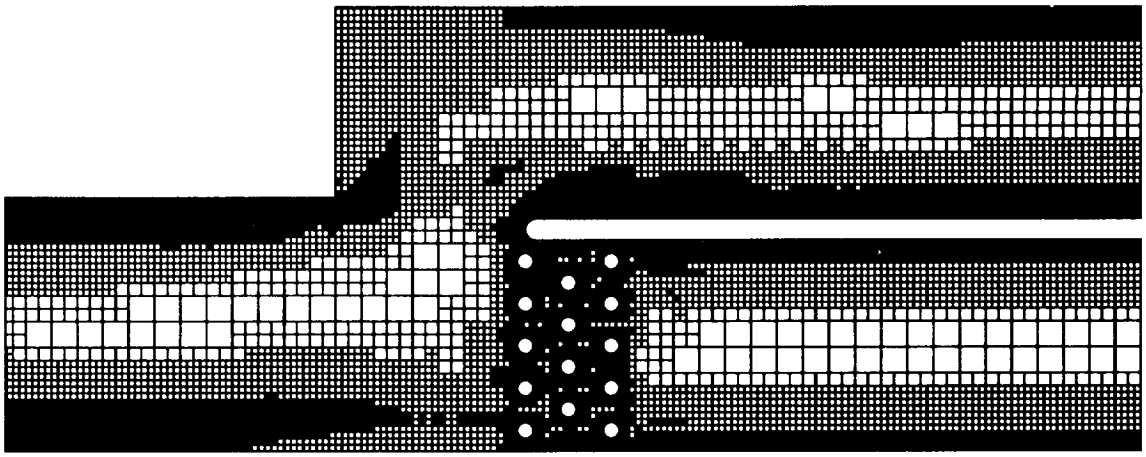


Figure 26 Close up of Final Adapted Grid, Lower Reynolds Number Case

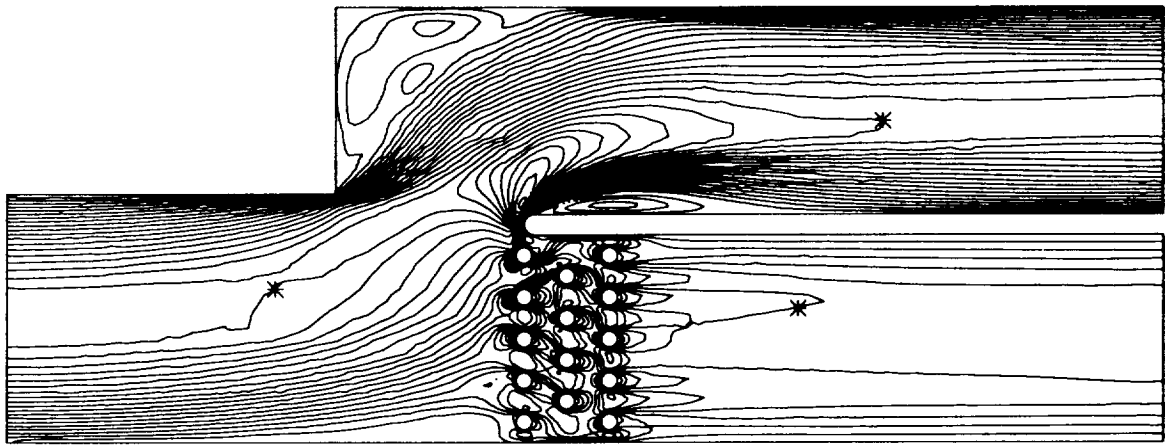


Figure 27 Close up of Final Grid Level Total-velocity Contours.

The basic flow features predicted here correspond to those in the experiment, although some important features are grossly under-resolved, such as the individual pin-fin wakes. The primary passage separation and reattachment along the splitter plate and the separation anchored at the back step portion are both properly predicted, as well as the upstream influence of the pin blockage upon the lower wall flow. Although many levels of refinement were not achieved, the larger scale flow features were adequately predicted and were improved by the mesh refinement procedure.

## V. CONCLUSIONS

An adaptively-refined, finite-volume solution procedure for the Euler and Navier-Stokes equations using a Cartesian, cell-based approach has been presented, demonstrated and validated. The Cartesian, cell-based grid generation procedure is automated, and is able to generate grids about complicated geometries without user intervention. The grid generation strategy is based upon the recursive, isotropic subdivision of a Car-

domain, polygonal cells are "cut" out of the background Cartesian mesh using a highly modified polygon clipping algorithm. The hierarchy of the grid generation process is stored in a binary tree, which provides a natural means of finding cell-to-cell connectivity (via logical tree traversals) and provides a straightforward means of adaptively refining the grid (via tree branch growth and pruning). Extension of this approach to the Navier-Stokes equations shows promise by providing a means of obtaining automated, adaptively-refined solutions upon domains where the grid has also been automatically generated.

A finite-volume, upwind-based scheme has been selected for treatment of the convective terms in the Euler/Navier-Stokes equations and has been implemented in the Cartesian, cell-based framework. Extension of the Cartesian approach for solving the Navier-Stokes equations has necessitated a careful investigation of candidate viscous flux formulations, which has been summarized here. The two viscous flux formulations that were investigated represent, respectively, a divergence theorem (Green-Gauss) based reconstruction procedure, and a quadratic polynomial based reconstruction procedure. Analysis and practice indicated that the Green-Gauss based procedure, where data at the face vertices was found in a linearity-preserving manner, was adequate and was also the more robust of the two, although neither could guarantee positivity on arbitrarily distorted meshes. This non-positivity reduced the robustness of the solver, although useful results can still often be obtained. The Green-Gauss based scheme, commonly referred to as the diamond-path reconstruction using a linearity-preserving weighting, has been used here to compute adaptively-refined solutions to a variety of low and moderate Reynolds number flows. Comparisons were made to accepted computational results, to experimental data and to theory for a range of flows, where the Cartesian, cell-based approach is shown to accurately predict these flows.

## ACKNOWLEDGEMENTS

For the multi-element airfoil, the authors gratefully acknowledge the ICASE workshop organizers for the geometry, Dr. Kyle Anderson of NASA LaRC for the Suddhoo-Hall reference and Dr. Darren DeZeeuw of The University of Michigan for the electronic form of the actual pressure coefficient data. The geometry for the coolant passage and the branched duct was graciously supplied by Dr. Erlendur Steinthorsson of ICOMP. The first author gratefully acknowledges the discussions with Dr. P.C.E. Jorgenson and Mr. C.J. Steffen, Jr..

## REFERENCES

[1] AGARD Subcommittee C. Test Cases for Inviscid Flow Field Methods. AGARD Advisory Report 211, 1986.

[2] B.F. Armaly, F. Durst, J.C.F. Pereira, and B. Schonung. Experimental and Theoretical Investigation of Backward-Facing Step Flow. *Journal of Fluid Mechanics*, 127:473–496, 1983.

[3] T. J. Barth and P. O. Frederickson. Higher Order Solution of the Euler Equations on Unstructured Grids Using Quadratic Reconstruction. AIAA Paper 90-0013, 1990.

[4] T.J. Barth and D.C. Jespersen. The Design and Application of Upwind Schemes on Unstructured Meshes. AIAA Paper 89-0366, 1989.

[5] S.A. Bayyuk, K.G. Powell, and B. van Leer. An Algorithm For the Simulation of 2-D Unsteady Inviscid Flows Around Arbitrarily Moving and Deforming Bodies of Arbitrary Geometry. AIAA 11th Computational Fluid Dynamics Conference Proceedings, 1993.

[6] M. J. Berger and P. Colella. Local Adaptive Mesh Refinement for Shock Hydrodynamics. Technical Report UCRL-97196, Lawrence Livermore National Laboratory, 1987.

[7] M.J. Berger. Adaptive mesh refinement for hyperbolic partial differential equations. *Journal of Computational Physics*, 53:484–512, 1984.

[8] M.J. Berger and R.J. LeVeque. An Adaptive Cartesian Mesh Algorithm for the Euler Equations in

Arbitrary Geometries. AIAA Paper 89-1930-CP, 1989.

[9] M.J. Berger and R.J. LeVeque. A Rotated Difference Scheme for Cartesian Grids in Complex Geometries. *Computing Systems in Engineering*, 1:305–311, 1990.

[10] Y.L. Chiang. *Simulation of Unsteady Inviscid Flow On An Adaptively Refined Cartesian Grid.* Ph.D. thesis, The University of Michigan, Department of Aerospace Engineering, 1992.

[11] D. K. Clarke, M. D. Salas, and H. A. Hassan. Euler Calculations for Multielement Airfoils Using Cartesian Grids. *AIAA Journal*, 24, 1986.

[12] W.J. Coirier. *An Adaptively-Refined, Cartesian, Cell-Based Scheme for the Euler and Navier-Stokes Equations.* Ph.D. thesis, The University of Michigan, Department Aerospace Engineering. Also published as NASA TM 106754, 1994.

[13] W.J. Coirier and K.G. Powell. An Accuracy Assessment of Cartesian-Mesh Approaches for the Euler Equations. AIAA Paper 93-3335-CP, 1993.

[14] D. DeZeeuw and K. G. Powell. An Adaptively-Refined Cartesian Mesh Solver for the Euler Equations. AIAA 10th Computational Fluid Dynamics Conference Proceedings, 1991.

[15] D. DeZeeuw and K.G. Powell. Euler Calculations of Axisymmetric Under-Expanded Jets by an Adaptive-Refinement Method. AIAA Paper 92-0321, 1992.

[16] D. DeZeeuw and K.G. Powell. An Adaptively Refined Cartesian Mesh Solver for the Euler Equations. *Journal of Computational Physics*, 104(1):56–58, 1993.

[17] D.L. DeZeeuw. *A Quadtree-Based Adaptively-Refined Cartesian-Grid Algorithm for Solution of the Euler Equations.* Ph.D. thesis, The University of Michigan, Department of Aerospace Engineering, 1993.

[18] U. Ghia, K.N. Ghia, and C.T. Shin. High-Re Solutions for Incompressible Flow Using the Navier-Stokes Equations and a Multigrid Method. *Journal of Computational Physics*, 48:387–411, 1982.

[19] C. Gooch and H. Oksuzoglu. Extension of State-Vector Splitting to the Navier-Stokes Equations. AIAA Paper 93-3374-CP, 1993.

[20] D.G. Holmes and S.D. Connell. Solution of the 2D Navier-Stokes Equations on Unstructured Adaptive Grids. AIAA Paper 89-1932-CP, 1989.

[21] M.S. Liou and C.J.Steffen, Jr. A New Flux Splitting Scheme. *Journal of Computational Physics*, 107:23–39, 1993.

[22] J. Melton, F. Enomoto, and M. Berger. Three-Dimensional Automatic Cartesian Grid Generation for Euler Flows. AIAA Paper 93-3386-CP, 1993.

[23] J.E. Melton, M.J. Berger, M.A. Aftosmis, and M.D. Wong. 3D Applications of a Cartesian Grid Euler Method. AIAA 33rd Aerospace Science Meeting, AIAA Paper 95-0853, 1995.

[24] C.R. Mitchell and R.W. Walters. K-Exact Reconstruction for the Navier-Stokes Equations on Arbitrary Grids. AIAA Paper 93-0536, 1993.

[25] R. Pember, J. Bell, P. Colella, W. Crutchfield, and M. Welcome. Adaptive Cartesian Grid Methods for Representing Geometry in Inviscid Compressible Flow. AIAA Paper 93-3385-CP, 1993.

[26] J.J. Quirk. *An Adaptive Grid Algorithm for Computational Shock Hydrodynamics.* Ph.D. thesis, Cranfield Institute of Technology, College of Aeronautics, 1991.

[27] J.J. Quirk. An Alternative to Unstructured Grids for Computing Gas Dynamic Flows Around Arbitrarily Complex Two-Dimensional Bodies. ICASE Report 92-7, 1992.

[28] L.M. Russell, D.R. Thurman, P.S. Simoni, S.A. Hippensteele, and P.E. Poinsatte. Measurements and Computational Analysis of Heat Transfer and Flow in a Simulated Turbine Blade Internal Cooling Passage. AIAA Paper AIAA-93-1797, 1993.

[29] P.H. Snynder and R.J.Roelke. The Design of an Air Cooled Metallic High Temperature Radial Turbine. AIAA Paper 88-2872, 1988.

[30] A. Suddhoo and I.M. Hall. Test Cases for the Plane Potential Flow past Multi-Element Aerofoils. *Aeronautical Journal*, pages 403–414, 1985.

[31] I.E. Sutherland and G.W. Hodgman. Reentrant Polygon Clipping. *Communications of the ACM, Graphics and Image Processing*, 17(1):32–42, 1974.

[32] G. Warren, W. K. Anderson, J. Thomas, and S. Krist. Grid Convergence for Adaptive Methods. AIAA 10th Computational Fluid Dynamics Conference Proceedings, 1991.

[33] D. P. Young, R. G. Melvin, M. B. Bieterman, and J. E. Bussoletti. A Locally Refined Rectangular Grid Finite Element Method: Application to Computational Fluid Dynamics and Computational Physics. *Journal of Computational Physics*, 92:1–66, 1991.

19951223 19

N95-28740

# DEVELOPMENT AND APPLICATION OF A 3D CARTESIAN GRID EULER METHOD

John E. Melton *
NASA Ames Research Center
Moffett Field, CA 94035

Marsha J. Berger †
Courant Institute
New York, NY 10012

Michael J. Aftosmis ‡
USAF/NASA Ames
Wright-Patterson AFB, OH 45433

Michael D. Wong §
Sterling Software
Palo Alto, CA 94305

## PREAMBLE

This paper documents the current state of development of our Cartesian non-body-fitted mesh algorithms for computing Euler flows around complex configurations. We include our most recent AIAA Aerospace Sciences Conference paper 95-0853 (ref. 1), which describes the geometric procedures used for the Cartesian grid generation. Two new appendices extend the description of the algorithms. Appendix I documents the flow solver used in all previous examples. Appendix II contains previously unpublished calculations for a supersonic missile (ref. 2).

## ABSTRACT

This report describes recent progress in the development and application of 3D Cartesian grid generation and Euler flow solution techniques. Improvements to flow field grid generation algorithms, geometry representations, and geometry refinement criteria are presented, including details of a procedure for correctly identifying and resolving extremely thin surface features. An initial implementation of automatic flow field refinement is also presented. Results for several 3D multi-component configurations are provided and discussed.

## INTRODUCTION

In this paper we discuss recent developments in our Cartesian grid approach to solving the Euler equations for flow around complex geometries. The Cartesian grid approach uses a non-body-fitted grid of rectangular cells to discretize the flow field about an object. After creating this surrounding mesh of rectangular cells, the surface geometry is simply "cut out" from the underlying cells, leaving a border of irregular cells surrounding the object. This approach has the potential to greatly simplify and automate the difficult task of grid generation around complex configurations. It replaces the difficult and case-specific problem of generating a body-fitted grid (structured, unstructured, multiblock etc.) with the more general problem of computing and characterizing the geometric intersections between the Cartesian flow field cells and the surface geometry. We present the algorithms used to automatically compute this required geometric information and describe some modifications to our original approach (ref. 3) that assist in the representation of very thin pieces of geometry, such as a spoiler, fin, or thin trailing edge section. Our algorithms have also been generalized so that the geometry of a complex configuration can be specified as a collection of distinct, repositionable, and possibly overlapping components; a single geometry description explicitly containing component intersections is no longer necessary.

---

* Aerospace Engineer, Applied Computational Aerodynamics Branch.
† Professor, Department of Computer Science.
‡ Aerospace Engineer, Technical Liaison Ames Research Center.
§ Technical Staff.

A Cartesian grid approach cannot be competitive unless it has the ability to vary the level of resolution according to the features of both the geometry and flow field. This is accomplished through the use of adaptive mesh refinement. A description of the criteria used to determine the level of refinement necessary to resolve a solid object is presented along with our approach for adaptively refining the mesh in response to features in the flow field. Finally, we demonstrate our approach by showing solutions for several three dimensional complex configurations and comparisons with experimental data.

Although various Cartesian-based approaches have been around for many years (cf. Ref. 4), renewed interest in these approaches has occurred due to continued difficulties with body-fitted grid generation. Some successful calculations in two and three dimensions, for both the potential and Euler equations, may be found in Refs. 5-12. These works differ in their approach to the data structures, flow solver, and adaptivity. Extensions to time dependent problems have been reported in Refs. 13-16.

## OVERVIEW OF CARTESIAN APPROACHES

Our Cartesian grid approach to solving the Euler equations is based on the finite volume form of the conservation equations:

$$\frac{d}{dt}\iiint_{Vol} w\,dx\,dy\,dz = -\oint_{S} \mathbf{f}\cdot\mathbf{n}\,dS \qquad \{1\}$$

where $w = (\rho, \rho u, \rho v, \rho w, \rho E)$ and

$$\mathbf{f}\cdot\mathbf{n} = \begin{pmatrix} \rho\mathbf{v}\cdot\mathbf{n} \\ \rho u\mathbf{v}\cdot\mathbf{n} + pnx \\ \rho v\mathbf{v}\cdot\mathbf{n} + pny \\ \rho w\mathbf{v}\cdot\mathbf{n} + pnz \\ (\rho E + p)\mathbf{v}\cdot\mathbf{n} \end{pmatrix} \qquad \{2\}$$

For the regular (non-intersected) grid cells off of the surface, any reasonable finite volume scheme can be used. In our case, this is a Jameson-Schmitt-Turkel scheme (ref. 17) using multi-stage Runge-Kutta integration and central differencing with added second- and fourth-order artificial viscosity. The scheme must however be modified to account for the details of the geometry at the irregular cells on the body surface. As Eq. 1 shows, the volumes and face areas of the irregular cells must be provided along with the normal vector for the piece of the surface within the Cartesian cell. For second-order schemes, the centroids of the cut cell volumes and faces are also required.

In previous work, we attempted to compute the geometric information directly from the NURBS (Non-Uniform Rational B-Spline) geometrical description of the object. This can be rather time-consuming, since simultaneous nonlinear equations must be solved to compute the intersection of a cell face with the surface geometry. Furthermore, the NURBS software package we chose was incapable of robustly performing the large number of intersection calculations required to make a high-resolution Cartesian grid. These experiences strongly influenced the development of our current grid generation approach.

Our current strategy separates the grid generation process into two distinct phases. In the first phase, and prior to any flow solutions, all possible cell geometric quantities that might be needed at a later stage (for example, during subsequent mesh refinement) are computed and saved in what is called the "database"

grid. This includes all the geometric information for the irregular cells at the finest possible level of refinement.

In the second phase of our grid generation process, the "actual" or "flow solution" grid is constructed solely from the geometric information stored in the database grid. This "flow solution" grid can have a variable level of resolution around the object; the appropriate level of refinement is determined by the surface curvature. The ultimate resolution of a flow solution grid is, however, limited to that of the database grid.

This separation of the grid generation process into two distinct phases and the creation of the database grid eliminates the need to repeatedly interrogate the surface geometry during later flow solution refinements. This guarantees that the flow solver will not abort midway to completion during a flow field refinement stage due to a failure in a geometric interrogation routine. However, because the database grid covers the entire surface geometry with cells of the finest resolution, our grid generation process has the disadvantages of requiring excessive storage and the calculation of possibly unnecessary geometric information.

The steps of the full 3D Cartesian grid flow simulation procedure are as follows:

**Step 1. Geometry Acquisition**

- Generate surface wireframe or triangulation files.

**Step 2. Cartesian Grid Generation**

*Database Grid Generation*

- Determine vertices of database grid cells.
- Compute exposed face areas of database cells.
- Sort database cells by x-location.

*Initial Flow Solution Grid Generation*

- Generate Cartesian flow solution grid using curvature and neighbor rule base.

**Step 3. Cartesian Flow Solutions**

- Cycle through flow solver and mesh adaptation steps until solution converges on grid with adequate resolution.

**Step 4. Postprocessing**

- Extract relevant flow field and surface data.

In the following sections we present the algorithmic details for the most important grid generation procedures, namely steps 2 and 3 above.

## Geometry Acquisition

The LaWGS (Langley Wireframe Grid Standard) format for surface geometry description has been adopted as the standard input format within the current method (ref. 18). Since all of the subsequent geometrical operations take advantage of the planar nature of triangles, the LaWGS networks used to define the individual components are decomposed into collections of triangles. A provision for reading surface geometry stored in conventional triangular finite element vertex and connectivity list formats is also available.

## Database Grid Generation

The following steps outline the creation of the surface-intersecting database grid cells.

1. Create coarse Cartesian background grid that encompasses the surface geometry.
2. Flag all cells that intersect the surface geometry.
3. Subdivide flagged cells, and eliminate those that are fully internal or fully external to the surface.
4. Recursively repeat Steps 2 and 3 until the maximum desired refinement level is reached.

Figure 1 depicts an extremely coarse database grid generated for a generic transport configuration. Note that the database cells can intersect several geometric components in the juncture regions between components.

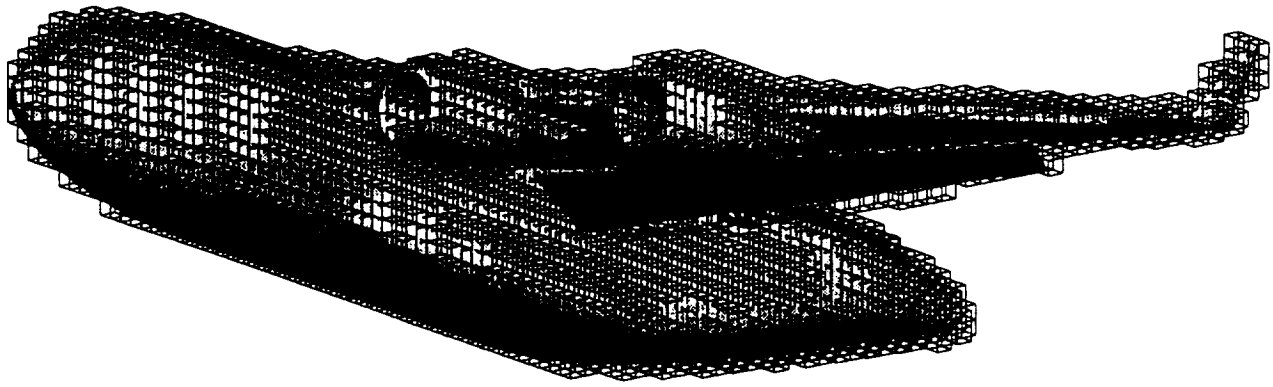


**Fig. 1. Database grid enveloping a generic advanced transport configuration. Grid shown is extremely coarse for clarity. Actual database grids for practical configurations are generally divided 4-5 levels beyond that shown.**

Face Intersection Calculations.—In Step 2 of the database grid generation process, each face of every cell must checked for intersection with the surface geometry components. For those components that pass an initial bounding-box test, the triangles that form the component must be checked for intersection with the face. In order for the face and triangle polygons to intersect, the intersection of one edge of one of the polygons must exist within the interior of the other. Figure 2 illustrates typical three-dimensional intersections between a triangle (polygon 1) and two non-coplanar rectangles (polygons 2 and 3).

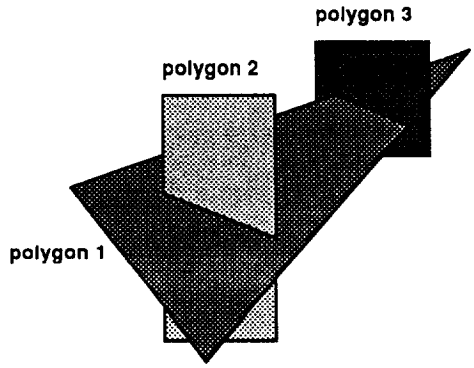



**Fig. 2. Typical polygon intersections.**

First, we compute the intersections between lines extending through each edge of polygon 1 and the plane that encompasses polygon 2. The polygons intersect along an edge of polygon 1 if the location of the intersection point lies both between the edge endpoints of polygon 1 and within the interior of polygon 2.

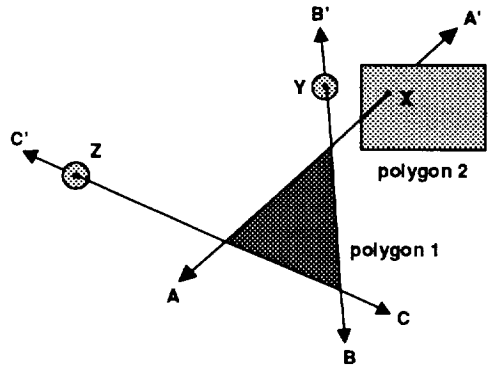



**Fig. 3. Computing intersections between polygon 1 and the plane containing polygon 2.**

In the case shown in Fig. 3, only the edge line *A-A'* of polygon 1 intersects the plane containing polygon 2 at point *X* within the interior of polygon 2. The *B-B'* and *C-C'* edge lines also intersect the plane of polygon 2, but both of these intersections (at *Y* and *Z*) lie outside polygon 2. Although the *A-A'* intersection lies within the interior of polygon 2, the intersection does not lie between the endpoints defining the edge of polygon 1; no intersection between polygons 1 and 2 exists.

In Fig. 4, the edge intersection line *B-B'* from polygon 1 intersects polygon 2 at point *X* within polygon 2 and between the edge endpoints; an intersection between polygon 1 and polygon 2 therefore exists.

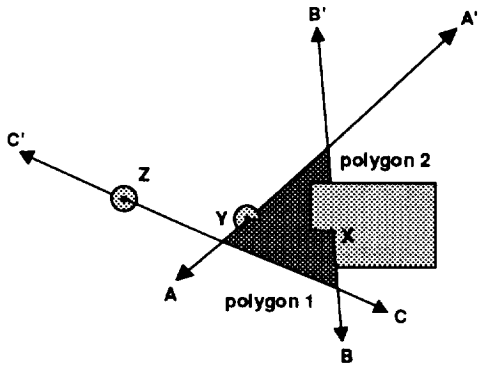



**Fig. 4. An intersection between polygon 1 and polygon 2 at X.**

Once the existence of intersections between the triangles that form a component and a cell face is determined, each intersection point must be characterized as occurring either interior to another component or on a flow field-exposed region of the surface. This is done using the two-step ray-casting algorithm illustrated in two dimensions in Fig. 5. First, a ray emanating from the point in question is cast in an arbitrary direction. Second, the number of intersections that occur between the ray and all other polyhedra is determined. If the ray intersects a polyhedron an even (or zero) number of times, then the point must lie outside of that polyhedron. If the number of intersections is odd, then the point lies within the polyhedron. Note that this procedure works for both concave and convex polyhedra. The polyhedra may also intersect and/or overlap, as long as each polyhedron is watertight (completely closed), with each edge of a polyhedral face shared by exactly one other facet of the same polyhedron.

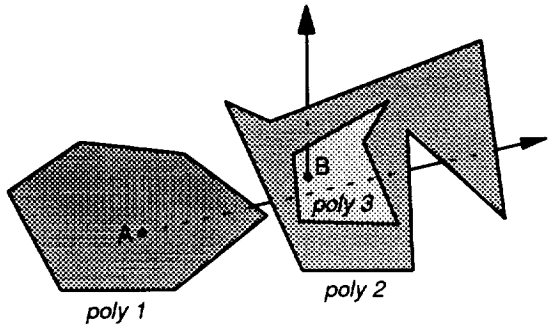



**Fig. 5. Inside/outside determination.**

In the above figure, the ray from point *A* makes one intersection with polygon 1, four intersections with polygon 2, and two intersections with polygon 3. Point *A* is therefore inside of polygon 1. A ray cast from point *B* makes zero intersections with polygon 1, one intersection with polygon 2, and one intersection with polygon 3. Point *B* is therefore inside of both polygon 2 and polygon 3. If no intersections with a surface component are found for a face, the ray-casting procedure is used to characterize the face as either fully internal or fully external to the surface geometry.

Face Area Calculations.—After the vertices of the database grid cells have been established, the areas of the exposed cell faces are computed. For each cell face, the cross section polygon is extracted from the surface component triangulation at the location coplanar with the cell face. The portion of this polygon that overlaps the cell face is then computed using the Sutherland-Hodgman polygon clipping algorithm (ref. 19). In Fig. 6, polygon 3 is the result of clipping the cross-section polygon 1 against the cell face polygon 2.

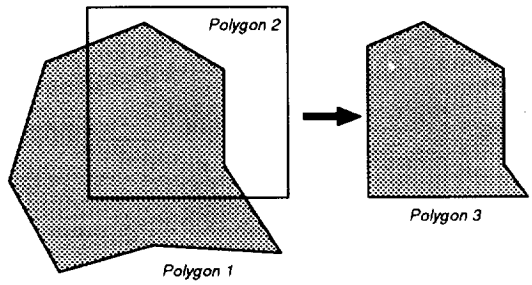



**Fig. 6. Polygon clipping.**

The amount of 'exposed' or 'flow-through' area for each face is then computed by subtracting the clipped cross-sectional area from the rectangular cell face area.

The procedure for computing the exposed database cell face areas becomes more complex when the surface geometry is composed of multiple (and possibly overlapping) components. Figure 7 illustrates the polygons resulting from a cross section slice at a location where the components do not intersect. The collection of polygons *A*, *B*, and *C* form the borders of polygon group *i*, and polygons *D* and *E* form the borders of polygon group *j*. Polygons *A* and *D* enclose 'solid' or 'non-flow' area, while polygons *B*, *C*, and *E* surround the 'flow-through' area within *A* and *D*.

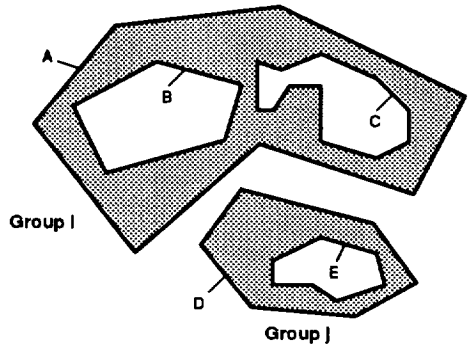



**Fig. 7. Polygon groups.**

For bodies composed of multiple intersecting components, the cross-section polygon groups arising from multiple components may overlap, as in Fig. 8.

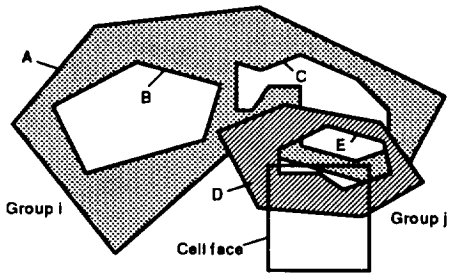



**Fig. 8. Overlapping polygon groups.**

To compute the net amount of each database cell face that is exposed or 'flow-through', an algorithm for computing the net amount of 'solid' area that overlaps the face rectangle is required. The equation for the solid area, $A^{-}_{total}$, can be expressed as

$$A^{-}_{total} = \sum_{i=1}^{n} A_i^{-} - \sum_{i=1}^{n} A_i^{+} + \sum_{i}^{n} \sum_{j}^{i \neq j} A_i^{-} \cap A_j^{+} - \frac{1}{2}\left( \sum_{i}^{n} \sum_{j}^{i \neq j} A_i^{+} \cap A_j^{+} + \sum_{i}^{n} \sum_{j}^{i \neq j} A_i^{-} \cap A_j^{-} \right) \qquad \{3\}$$

In Eq. 3, the indices $i$ and $j$ refer to individual component 'polygon groups', where $A$ is the area of the polygon group contained within the rectangular boundaries of the face. The '+' and '-' superscripts denote the external (flow-through) and internal (to a component) areas enclosed by each group, respectively. The input to the overlap algorithm requires a listing of all the polygons associated with each group and that all polygons within every group be initially designated as enclosing either internal or external area. Finally, the Sutherland-Hodgman algorithm is used to compute the net amount of overlapping area contributed by the polygon groups. The use of the Sutherland-Hodgman algorithm is complicated by the fact that the target clipping polygon must be convex, so the Boolean intersection of two possibly non-convex polygons must be computed by first decomposing each polygon into two sets of triangles, and then keeping track of the sum of the areas resulting from clipping all triangles in each polygon against those of the other polygon.

This adoption of a component-based approach to Cartesian grid generation is somewhat analogous to the use of Chimera procedures for conventional body-fitted structured grids (ref. 20). Both techniques are intended to eliminate the need to recreate new surface definitions reflecting updated intersection information

whenever individual components are translated or rotated. Since the database grid cells used in our procedure are restricted to the exterior surface of intersecting components, the intersection regions between otherwise smooth components are automatically detected. The geometric intersection information computed and saved for each cell in the database grid consists of the cell vertex coordinates and the flow field-exposed area of each cell face.

## Flow Solution Grid Generation

The second phase of the grid generation process uses information extracted from the database grid to build an initial flow field mesh. Starting from a coarse background grid, the Cartesian cells are automatically refined, based only on the curvature of the geometry, until the appropriate resolution is reached. This initial mesh is then used to begin the flow solutions. Further mesh refinements, based on the flow solution itself, occur during later flow field refinement stages.

The automatic refinement of the initial mesh is based on the surface curvature, which in turn is derived from the local variation of the surface normal vectors. Although the surface normal vector within each database grid cell is not explicitly included in the database, its components can be computed by taking the difference in the exposed areas of opposing faces. This procedure is illustrated in two dimensions in Fig. 9.

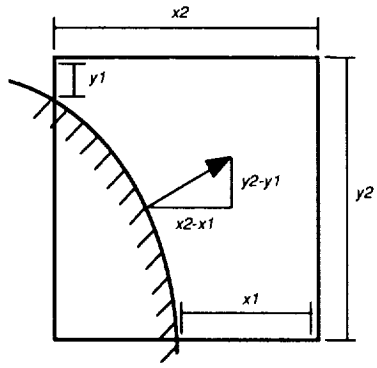



**Fig. 9. Representative surface normal computed using difference in opposing exposed face areas.**

For irregular flow field cells, the necessary surface normal vector and face areas are also easily obtained by summing the face areas of encompassed database cells. Figure 10 shows a flow field cell and its encompassed database cells.

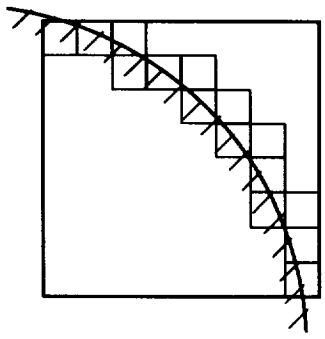


**Fig. 10. Flow cell with encompassed database cells.**

We use two curvature rules to refine cells in the initial flow field mesh. The first rule is based on the maximum difference in the surface normal direction between neighboring flow field cells. If the difference exceeds a user-specified tolerance, the cell is marked for additional refinement. The second rule uses the finer information in the database grid itself. If the maximum angular difference in the surface normals of the encompassed database cells exceeds a second user-specified tolerance, the flow field cell is marked for refinement, even if it passed the first test. Using these rules for the automatic refinement of the initial flow field mesh has two benefits. First, the magnitude of the surface curvature and the appropriate level of grid refinement are not inferred solely from the size of the elements used to discretize the surface. Second, the intersection regions between overlapping components are automatically detected and appropriately refined.

After the cells have been tested and marked for refinement, a buffer layer of neighboring cells is added. This creates smooth transition regions between refinement levels. Under no circumstances are neighboring cells allowed to differ by more than one level of refinement. Typical values for our grid generation control parameters are provided in Table 1.

| | |
|---|---|
| Database angle difference | 40° |
| Flow field angle difference | 40° |
| Number of neighbor buffer layers | 3 |
| Total number of refinement passes | 7-12 |

**Table 1. Typical values used to create initial flow field grids.**

## Multiple Region Algorithms

One major disadvantage of the Cartesian approach and its reliance on a non-body-fitted grid is the difficulty in providing sufficient resolution for extremely thin components. To accurately represent the surface geometry of thin components (such as wing leading and trailing edges, fins, and engine cowlings), an excessive number of highly-refined cells are often required. Once a cell that has been split by the geometry into multiple independent regions has been identified, it should instead be replaced by multiple, non-isotropic new cells having the correct face areas and connectivity. A method of detecting cells that have been split into two or more distinct regions is therefore highly desirable, and an overview of our approach follows.

Figure 11 illustrates the split-cell situation in two dimensions:

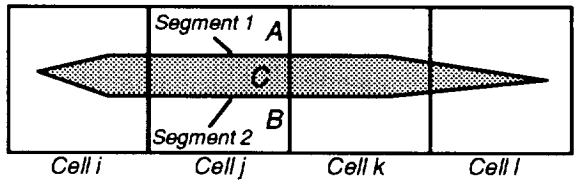



**Fig. 11. Cells split into distinct regions.**

Cell $j$ is obviously split into three distinct polygonal regions (labeled $A$, $B$, and $C$). Polygons $A$ and $B$ are external to the geometry, while polygon $C$ is completely internal. These polygons are identified and created in a procedure that examines the two segments that form the outline of the portion of the airfoil within cell $j$ and categorizes their intersections with the cell edges (in the above figure, the segments are labeled Segment 1 and 2). Once the edges of each polygon region have been established, the area within

each polygon is denoted as either fully internal or fully external to the geometry. The three-dimensional procedure for determining the number of independent regions into which a cell has been split begins by applying this two-dimensional procedure to each face and the segments resulting from the intersection of the surface geometry with that face.

Figure 12 depicts two adjoining cells (separated for clarity) intersected by the trailing edge of a wing. The portions of the wing surface within the cells are shaded.

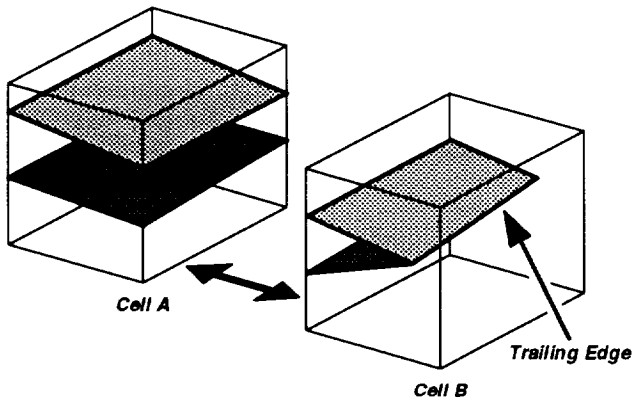



**Fig. 12. Two cells near wing trailing edge.**

Figure 13 shows an exploded view of the individual polygons that compose the faces of the cells.

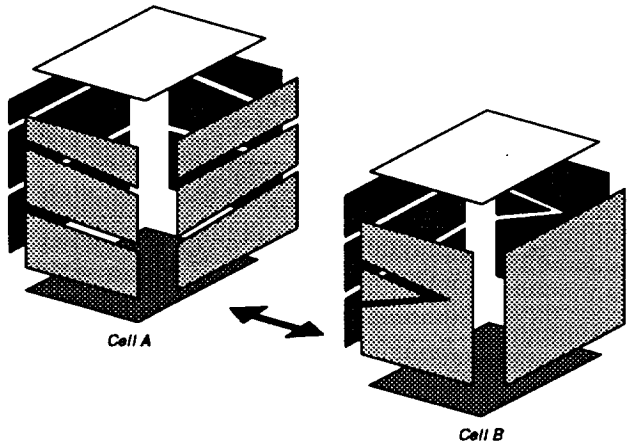



**Fig. 13. Exploded view of face polygons.**

After the independent polygons that compose each face are determined, matches between polygon edge segments that lie along the edges of the parent Cartesian cell are used to group adjoining polygons into the face collections that will compose the new independent regions. These links are shown with symbolic shading in Fig. 14

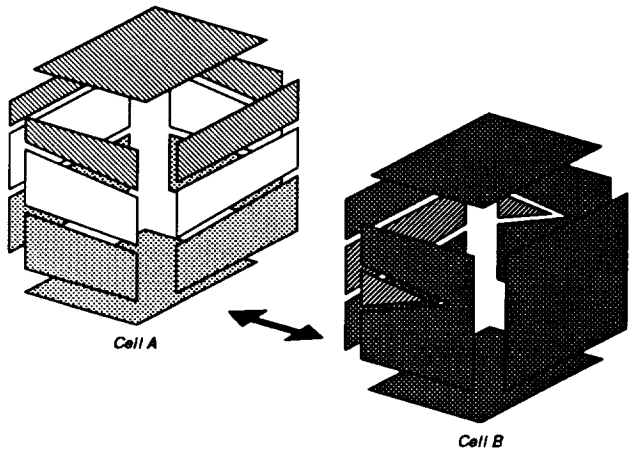



**Fig. 14. Symbolic links between matching edges.**

The portions of the surface that lie within the cell are then similarly matched and included into the face collections to form the complete face description of the independent regions. The exploded view of the independent regions is presented in Fig. 15, which shows that Cell *A* was split into three distinct regions, while Cell *B* was split into two.

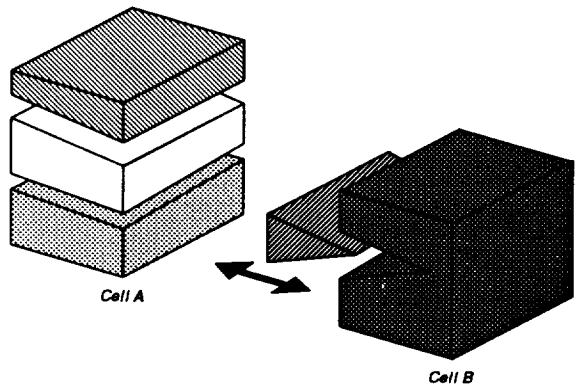



**Fig. 15. Exploded view of distinct cell regions.**

Using the faces that compose each polygon, the face and volume centroids of the resultant new individual regions can be computed, and the combined volumes and centroids summed and checked for consistency with the parent cell.

## Volume and Centroid Calculations

The cell volumes and centroids are calculated using the Gauss divergence theorem, which transforms the required volume integrations into surface integrations over the exposed and intercepted faces of each cell. After decomposing each face polygon into triangles, a third-order three-point quadrature rule is used for the integrations.

## SOLUTION ADAPTIVE CELL DIVISION

With the cell division and surface interrogation algorithms already developed for the geometric based cell refinement, the addition of solution adaptation is relatively straightforward. The approach adopted in the current work follows that presented by Aftosmis (ref. 21), appropriately modified for use within the Cartesian framework.

### Approach

The underlying methodology may be characterized as a "feature detection" algorithm, since it operates directly on computed flow field differences, rather than strict discretization error estimates within the discrete solutions. The adaptation scheme uses a two pass approach in scanning the flow field for cell division candidates. The first path searches for shocks or sharp expansions, while the second tags cells containing smooth inviscid features.

This approach is based upon the observation that in a general flow field, shocks of varying strength may cause many parameters to jump by orders of magnitude (e.g. static pressure through a strong shock). Thus, there exists a danger of overlooking less prominent features with a crude detection algorithm. Failure to adapt the mesh to capture such "smooth features" may actually lead to an adaptive procedure which converges to the wrong solution (ref. 22).

### Shock and Smooth Feature Detection

A normalized, undivided second difference of pressure provides a parameter which is quite sensitive to both strong and weak shocks. This quantity recognizes curvature (in computational space) in the pressure profile across a difference stencil spanning three cells. The Cartesian components of a directional refinement parameter $\mathbf{R}_s$ may be defined using a stencil that extends into the six nearest neighbors of each cell:

$$\begin{aligned} \mathbf{R}_S &= R_{S_x}\mathbf{i} + R_{S_y}\mathbf{j} + R_{S_z}\mathbf{k} \\ &= \frac{\delta_x^2 p}{\overline{p}_{(x)}}\mathbf{i} + \frac{\delta_y^2 p}{\overline{p}_{(y)}}\mathbf{j} + \frac{\delta_z^2 p}{\overline{p}_{(z)}}\mathbf{k} \end{aligned} \tag{4}$$

where $\delta_x^2 p = p_{i-1,j,k} - 2p_{i,j,k} + p_{i+1,j,k}$

and $\overline{p}_{(x)} = p_{i-1,j,k} + 2p_{i,j,k} + p_{i+1,j,k}$

Cells are tagged for division when the vector magnitude of $\mathbf{R}_s$ exceeds a preset absolute threshold $T_s$ (usually set to 0.05).

$$\left|\mathbf{R}_S\right| > T_S \tag{5}$$

With the cells spanning strong nonlinearities identified, only the remaining cells are re-scanned for smoother features. Since the extrema have been removed, the remaining discrete data may be considered smoothly varying, and the refinement proceeds on a statistical basis as described in Ref. 23.

The parameter chosen to identify smooth, inviscid features in this preliminary work uses undivided first differences of density, and is computed directionally as:

$$\mathbf{R}_I = \frac{\delta_x \rho}{\overline{\rho}_{(x)}}\mathbf{i} + \frac{\delta_y \rho}{\overline{\rho}_{(y)}}\mathbf{j} + \frac{\delta_z \rho}{\overline{\rho}_{(z)}}\mathbf{k} \qquad (6)$$

Cells are tagged for division if the magnitude of $\mathbf{R}_I$ exceeds the flow field mean by a pre-specified fraction, $T_I$, of the standard deviation, $\sigma$.

$$\text{if } \left|\mathbf{R}_I\right| > \overline{\left|\mathbf{R}_I\right|} + \sigma T_I \quad \text{then tag cell for division}$$

$T_I$ was set to 0.1 for the adaptation example in the next section. Ellipsoidal and hexahedral selection tools permit the user to restrict adaptation to certain regions of the flow field. This ability further focuses computational resources on specific regions of an evolving discrete solution.

## CARTESIAN APPLICATIONS

Selected results for three different configurations are presented in the following sections. All of the applications were performed on the CRAY C-90 at the Numerical Aerodynamic Simulation (NAS) facility at NASA Ames Research Center.

### Advanced High Wing Transport

Aircraft Figure 16 displays a front view of an advanced, high wing, transport configuration. The model is a four engine aircraft and includes the fuselage, wing, engine pylons, nacelles and mass-flow plugs within each nacelle. Symmetry permitted use of a half model in the simulations. The surface description of this model consisted of twelve LaWGS networks, each of which was triangulated to form a closed, watertight polyhedron. Intersection regions between overlapping polyhedra were successfully identified and automatically discretized within the database grid generation process. The aircraft was simulated in transonic cruise at a low angle of attack.

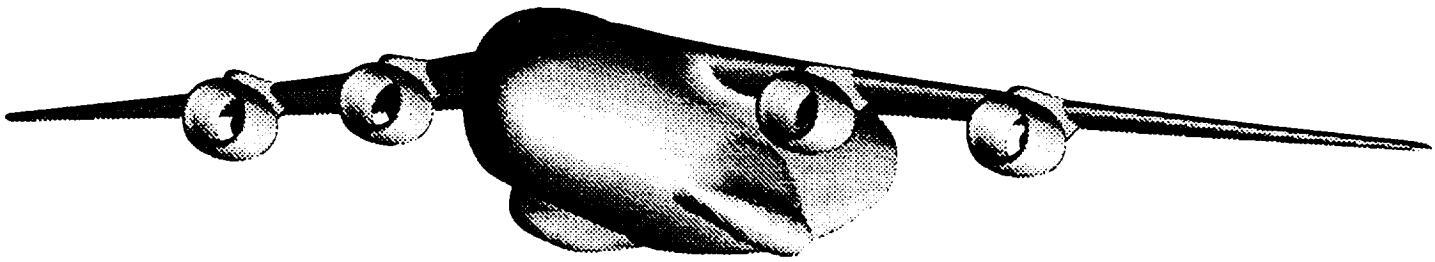


**Fig. 16. Front view of an advanced, high-wing transport configuration with fuselage, wing, engine pylons, engine nacelles, and mass flow plugs defined by 12 intersecting and overlapping surface triangulations.**

Starting from an initial 32x32x16 background grid, the solution process began with the generation of a five-level, geometry-adapted mesh. This coarse initial grid contained approximately 70,000 cells, and is shown in Fig. 17.

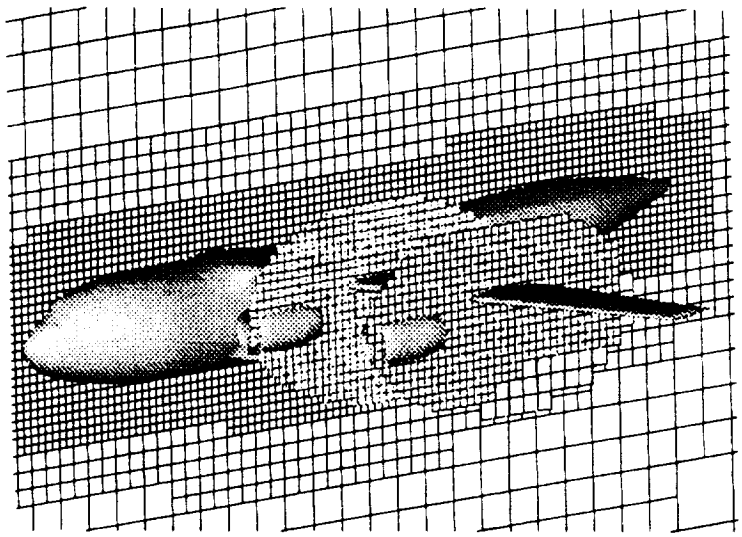


**Fig. 17. Geometry-adapted starting mesh.**

Note that since this mesh was created solely through the geometric cell refinement procedure described in section II.C, it actually constitutes the "starting mesh" for the adaptive flow solution. Using this grid as a starting point, the methodology generated a final solution by cycling the discrete solution through five successive flow solver - adaptive mesh refinement stages. The final mesh contained approximately 2.9 million nodes, ten levels of refinement, and extended 20 fuselage lengths fore and aft, above and below, and 10 fuselage lengths spanwise from the geometry. Figure 18 provides a view of the final, solution adapted mesh. The multiple region technology was not used in the calculations.

Figure 19 displays two views of the isobars in the discrete solution. The figure shows the presence of the main wing shock as it extends up and over the fuselage. The solution is mapped onto cutting planes at two wing stations aligned with the center of each engine nacelle. The resolution of the shock can be seen both on the main wing and inside the flow-through nacelles. Figure 20 shows a close-up of the wing shock and the mesh adaptation pattern between the inboard pylon and the fuselage. This figure also shows some irregularities in the mesh adaptation which will be removed through improvements in the rule base governing the adaptation. Surface isobars and selected spanwise $CP$ cuts for this case were also computed and compared well with experimental data.

## High Speed Civil Transport

Figure 21 shows the surface pressure distribution for an advanced supersonic transport configuration computed with the Cartesian method. Two grids consisting of 260,000 and 490,000 cells were also generated for the configuration without nacelles and diverters. Extensive calculations and comparisons with experimental data for the wing-body configuration were made at Mach numbers from 1.65 through 2.4. In Fig. 22, the predictions at Mach 1.8 of the lift, drag, and pitching moment coefficients computed using the two grids are compared with wind tunnel data. Experimental comparisons of lift, drag and pitching moment coefficient versus angle of attack were in similar agreement for all other Mach numbers. The multiple region technology was not used in the calculations.

## Wing C

An initial demonstration of the multiple-region cell technology was performed using the Wing C configuration (ref. 24). This wing has a leading edge sweep of 45 degrees, an aspect ratio of 2.6, and a thin, non-planar trailing edge due to its use of twisted, supercritical airfoil sections. Two grids of approximately 203,000 and 610,000 cells were used to compute transonic flow solutions. The 610,000 cell grid was created by performing one additional refinement of the coarse grid. Both coarse grid and fine grids were generated in a total of 30 C-90 CPU minutes. The dimension of the smallest cells in the finest grid is

approximately 2.8 times larger than the wing trailing edge thickness, and 0.0063 of the mean aerodynamic chord. After computing solutions at a Mach number of 0.85 and an angle of attack of 5 degrees for both the coarse and fine grids, the coarse grid was examined for cells that could be split into multiple distinct regions. Using the multiple-region algorithms, new cells were then added, yielding a corrected version of the coarse grid with approximately 204,000 cells. The flow solver executed at a rate of 6.5 $\mu$-seconds/cell/iteration, and all solutions were converged at least 4 orders of magnitude in the density residual. Solutions for all three grids are compared to wind tunnel data at selected span stations in Fig. 23. Most of the disagreement between the computed and experimental distributions at the outboard station can be attributed to viscous effects and the separation cell observed in the wind tunnel tests, but these differences were not quantified. The advantages of the multiple-region technology are, however, clearly evident. Without the multiple-region technology, the distinction between upper and lower surface cells at the trailing edge is lost, and the effective local chord for the coarse uncorrected grid is shortened by approximately ten percent. Although the fine grid results show that this effect can be reduced by doubling the cell resolution, the application of the multiple-region algorithms is clearly a significantly more efficient means of capturing extremely thin geometric details. Figure 24 shows the upper surface of the Wing C configuration colored by the computed surface pressure distributions for the coarse grids. The effect of the uncorrected multiple-regions cells along the trailing edge is clearly evident in the left half of the figure. The right half of the figure indicates the benefits of the application of the multi-region technology and the associated reduction in the surface pressure errors.

## CONCLUSIONS

The current grid generation approach has been used to develop and successfully demonstrate four important features necessary for future Cartesian grid approaches. The first is the capability to identify and refine flow field cells in regions of large surface curvature before a single flow solution is performed. The second feature is a "component-based" capability for handling surface geometry. This feature allows simple translations and rotations to be applied to individual component definitions to create new configuration variations, eliminating the need to return to the CAD system to create new surface discretizations that explicitly reflect the intersections between components. Third, a means for correctly distinguishing the upper and lower surface of extremely thin (less than a cell dimension) geometrical features improves the fidelity of the simulations without excessive refinement. The fourth feature is the full incorporation of automatic and adaptive flow field mesh refinement. Current work focuses on improving the accuracy of the surface boundary conditions and restructuring the grid generation procedures for increased efficiency.

This research has targeted the development and integration of many of the important technologies required for a future automated Euler Cartesian CFD procedure. Demonstrations of their application to arbitrary, three-dimensional configurations indicate the method's ability to dramatically reduce the time and effort required to produce useful inviscid CFD simulations for complex vehicles.

## ACKNOWLEDGMENTS

M. Berger was supported in part by DOE grant DE-FG02-88ER25053, and by AFOSR grant F49620-94-1-0132. NASA's Numerical Aerodynamic Simulation Facility provided much of the computer time used in this study. This support is gratefully acknowledged.

## APPENDIX I

This appendix contains additional details about the flow solver used to compute the examples in this paper. The flow solver is currently undergoing vigorous development, with special attention focused on the procedures at the solid wall boundary. We include here a description of our current algorithm for documentation purposes. The flow solver uses a multistage Jameson-Schmitt-Turkel (ref. 17) Runge-Kutta time integration procedure with local time-stepping to reach steady state (a multigrid acceleration scheme has not yet been implemented). For all interior points in the domain (e.g., those whose stencil does not contain a cut cell next to the solid wall), the usual central difference scheme augmented with pressure-switched second- and fourth-order dissipation terms is applied. Typical values of the user-specified second- and fourth-order dissipation coefficients are 1.2 and 0.5, respectively. At the edge of the computational domain, these discretizations are applied on cells that are no longer regular. Currently, the location of the volume centroid and the area centroids of the exposed portions of cut faces are not used when computing the fluxes. Along the boundaries, the order of accuracy of the basic central difference scheme therefore drops to first order. For cells with cut faces or faces completely internal to the geometry, the second and fourth derivative dissipation terms are contaminated by first and third difference contributions. Numerical experiments have shown that the errors introduced by these dissipation terms appear, however, to be much smaller than those associated with our current simple flux calculation strategy and our first-order solid wall boundary condition, in which the pressure at the wall is extrapolated from the interior using piecewise constant extrapolation. These cut cell discretizations have been improved in research versions of the code, but are not yet incorporated into the production version. In careful numerical studies of these methods (which will be reported elsewhere), improved strategies for the flux calculations at intersected faces and along the solid wall boundaries lead to a considerable reduction in the boundary error.

## APPENDIX II

Static aerodynamic coefficients for a supersonic missile configuration (Ref. 2) were computed at Mach numbers between 2.30 and 4.63. The missile has an ogive nose cone and four cruciform fins mounted at the aft end of a cylindrical fuselage (side and front views are provided in Fig. 25, which is taken from Ref. 2). Three grids of 290,000, 470,000, and 624,000 cells were used. The lift, drag, and pitching moment coefficients for all three grids are compared with experimental data for Mach 2.30 in Figure 26. Additional comparisons with the 290,000 cell grid for Mach numbers 2.96, 3.95, and 4.63 are presented in Figures 27, 28, and 29. The trends with increasing Mach number are seen to be accurately predicted, and the effects of changes in angle of attack are in good agreement up to approximately 15 degrees. No corrections for skin friction were made to the computed drag values.

## REFERENCES


1. Melton, J.E., Berger, M.J., Aftosmis, M.A., and Wong, M.J., "3D Applications of a Cartesian Grid Euler Method," AIAA Paper 95-0853, January 1995.
2. Fuller, D.E., Richardson, C.S., and Spearman, M.L., "Aerodynamic Characteristics of a Wingless Missile with Aft Cruciform Controls at Mach Numbers from 2.30 to 4.63," NASA TM-X 2488, 1972.
3. Melton, J.E., Enomoto, F.Y., and Berger, M.J., "3D Automatic Cartesian Grid Generation for Euler Flows," AIAA Paper 93-3386-CP, July 1993.
4. Varga, R., *Matrix Iterative Analysis*, Prentice Hall, Englewood Cliffs, N.J., 1962.

5. Purvis, J.W., and Burkhalter, J.E., "Prediction of Critical Mach Number for Store Configurations," *AIAA Journal*, Vol. 17, No. 11, November 1979, pp. 1170-1177.

6. Samant, S.S., Bussoletti, J.E., Johnson, F.T., Burkhart, R.H., Everson, B.L., Melvin, R.G., Young, D.P., Erickson, L.L., and Madson, M.D., "TRANAIR: A Computer Code for Transonic Analyses of Arbitrary Configurations," AIAA Paper 87-0034, January 1987.

7. Gaffney, R.L., Hassan, H.A., and Salas, M.D., "Euler Calculations for Wings Using Cartesian Grids," AIAA Paper 87-0356, January 1987.

8. De Zeeuw, D., and Powell, K.G., "An Adaptively-Refined Cartesian Mesh Solver for the Euler Equations," AIAA Paper 91-1542-CP, April 1991.

9. Tidd, D.M., Strash, D.J., Epstein, B., Luntz, A., Nachson, A., and Rubin, T., "Application of an Efficient 3-D Multigrid Euler Method (MGAERO) to Complete Aircraft Configurations," AIAA Paper 91-3236, September 1991.

10. Morinishi, K., "A Finite Difference Solution of the Euler Equations on Non-body-fitted Cartesian Grids," *Computers Fluids*, Vol. 21, No. 3, 1992, pp. 331-344.

11. Coirier, W.J., "An Adaptively-Refined, Cartesian, Cell-Based Scheme for the Euler and Navier-Stokes Equations," NASA TM 106754, October 1994.

12. Karman, S.L., "Splitflow: A 3D Unstructured Cartesian/Prismatic CFD Code for Complex Geometries," AIAA Paper 95-0343, January 1995.

13. Berger, M.J., and LeVeque, R.J., "An Adaptive Cartesian Mesh Algorithm for the Euler Equations in Arbitrary Geometries," AIAA Paper 89-1930-CP, June 1989.

14. Berger, M.J., and LeVeque, R.J., "Stable Boundary Conditions for Cartesian Grid Calculations," ICASE Report No. 90-37, May 1990.

15. Quirk, J.J., "An Alternative to Unstructured Grids for Computing Gas Dynamic Flows Around Arbitrarily Complex Two-Dimensional Bodies," ICASE Report 92-7, February 1992.

16. Pember, R.B., Bell, J.B., Colella, P., Crutchfield, W.Y., and Welcome, M.L., "Adaptive Cartesian Grid Methods for Representing Geometry in Inviscid Compressible Flow," AIAA Paper 93-3385-CP, July 1993.

17. Jameson, A., Schmidt, W., and Turkel, E., "Numerical Solutions of the Euler Equations by Finite Volume Methods using Runge-Kutta Time-Stepping Schemes," AIAA-81-1259, June 1981.

18. Craigdon, C. B., "A Description of the Langley Wireframe Geometry Standard (LaWGS) Format," NASA TM 85767, February 1985.

19. Foley, J.D., van Dam, A., Feiner, S.K., and Hughes, J.F., *Computer Graphics, Principles and Practice*, Addison-Wesley, 1990.

20. Benek, J.A., Buning, P., and Steger, J.L., "A 3-D Chimera Grid Embedding Technique," AIAA Paper 85-1523, July 1985.

21. Aftosmis, M.J., "Upwind Method for Simulation of Viscous Flow on Adaptively Refined Meshes," *AIAA Journal*, Vol. 32, No. 2, February 1994, pp. 268-277.

22. Warren, G.P., Anderson, W.K., Thomas, J.L., and Krist, S., "Grid Convergence for Adaptive Methods," AIAA Paper 91-1592, June 1991.

23. Kallinderis, Y.G., "Adaptation Methods for Viscous Flows," Doctoral Dissertation, Massachusetts Institute of Technology, Cambridge, MA, 1988.

24. Keener, E.R., "Pressure-Distribution Measurements on a Transonic Low-Aspect Ratio Wing," NASA TM 86683, September 1985.

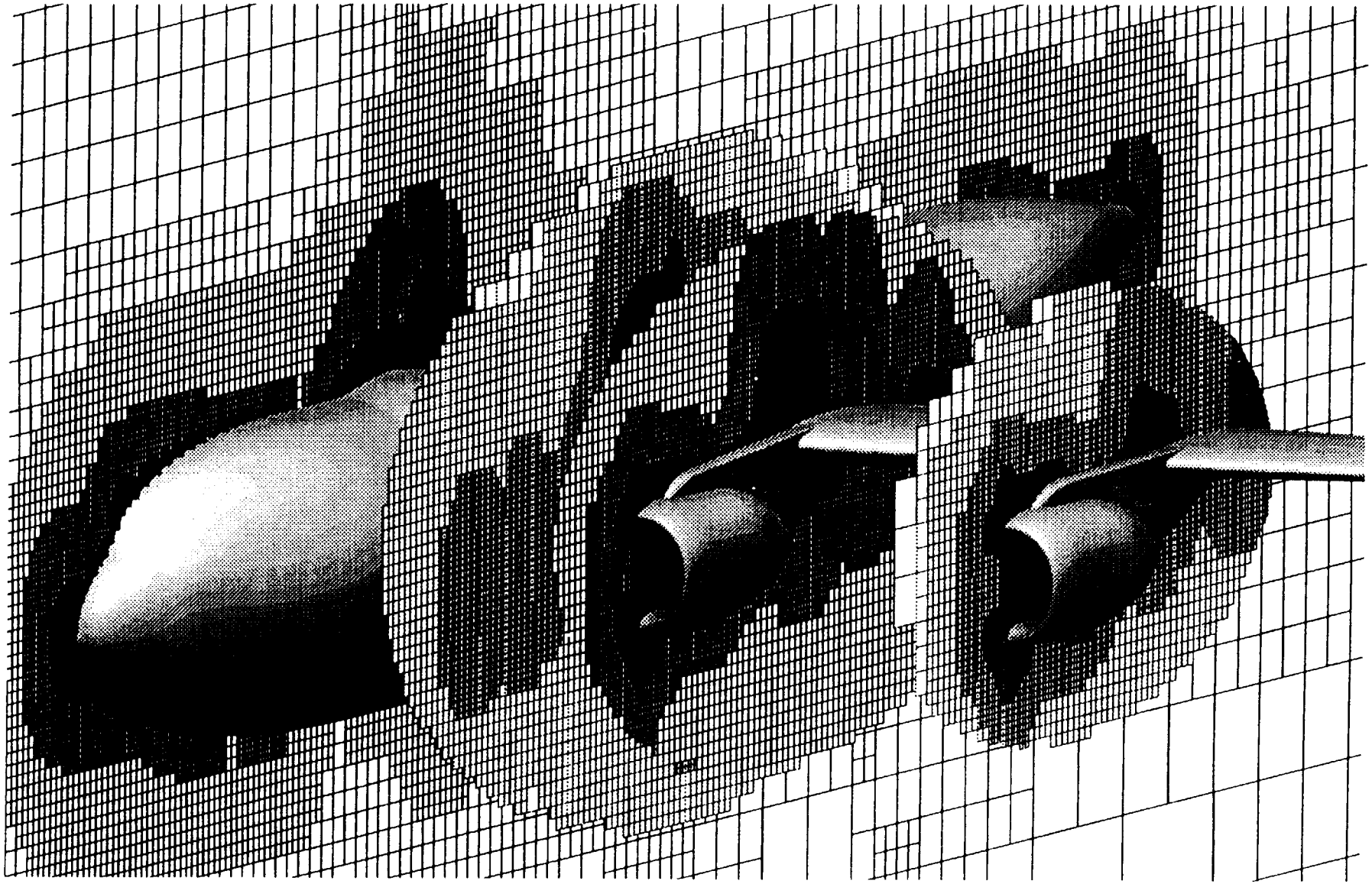


**Figure 18. Adapted computational mesh for the advanced high-wing transport. The mesh contains over 2.9 million nodes. The aircraft surface is defined by twelve separate surface triangulations.**

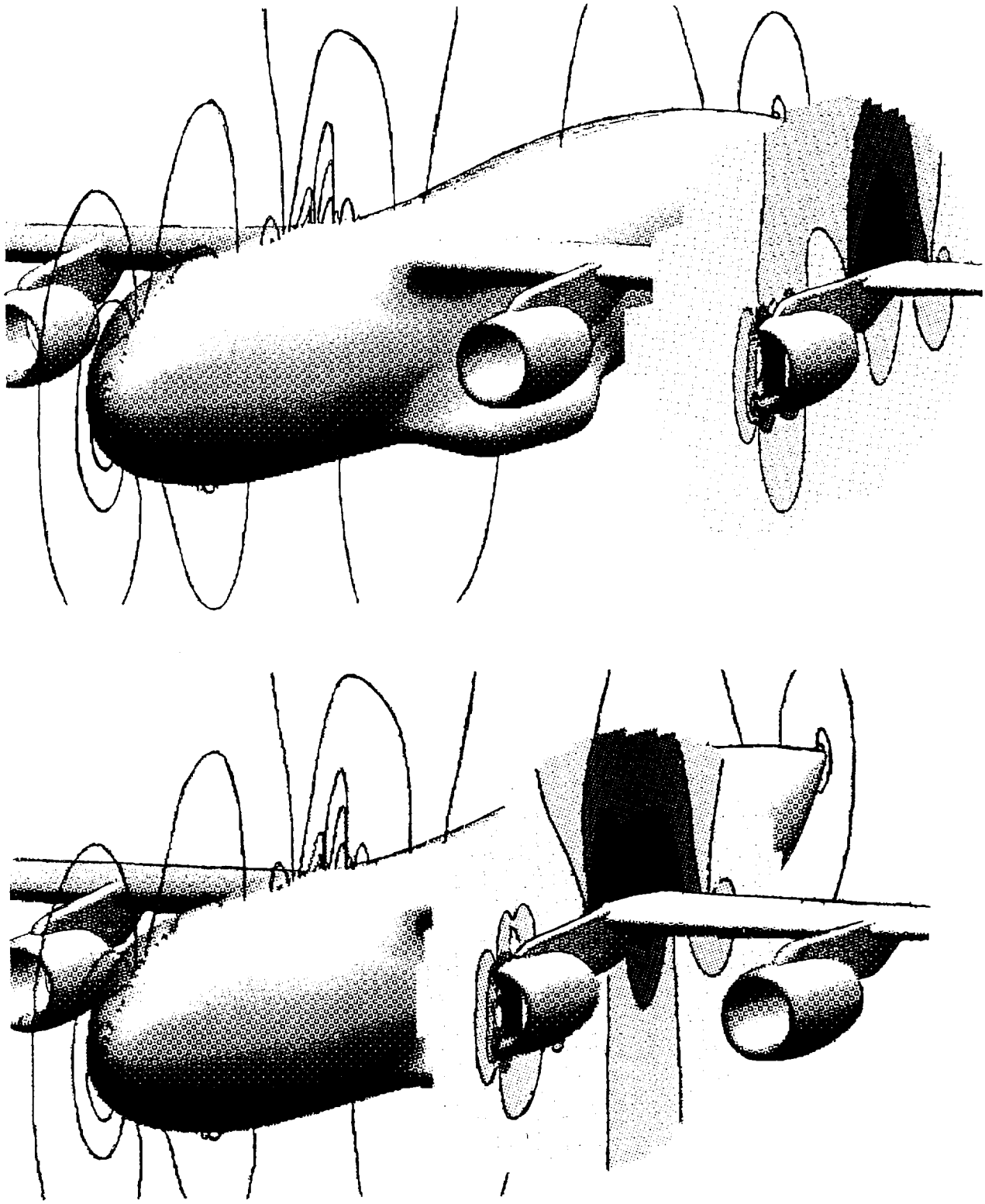


Figure 19. Pressure contours for the advanced high-wing transport. Contours are displayed in the symmetry plane and on cutting planes aligned with the inboard and outboard nacelles.

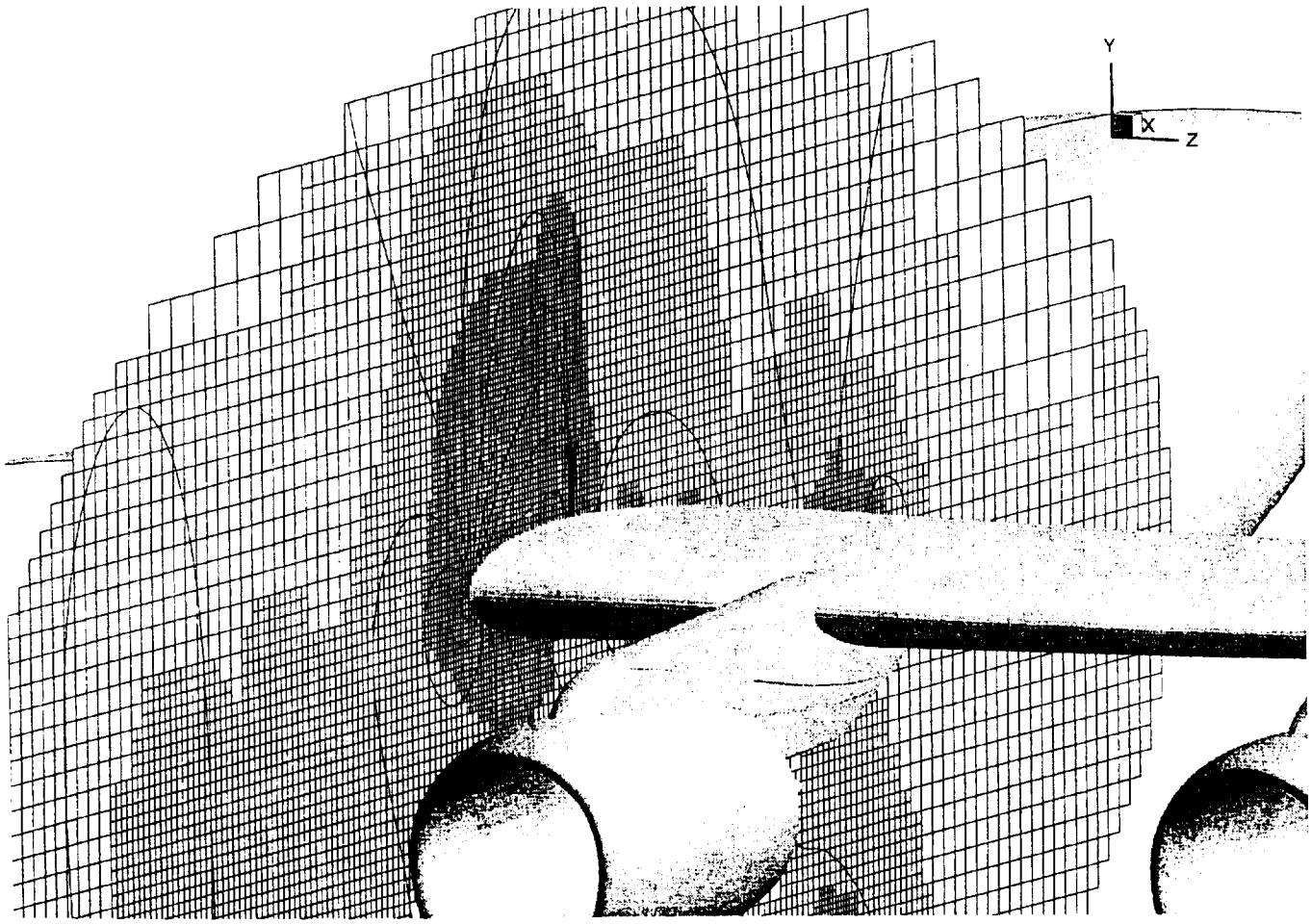



Figure 20. Close-up of wing station near inboard nacelle showing final adapted mesh and flow field pressure contours for advanced high-wing transport.

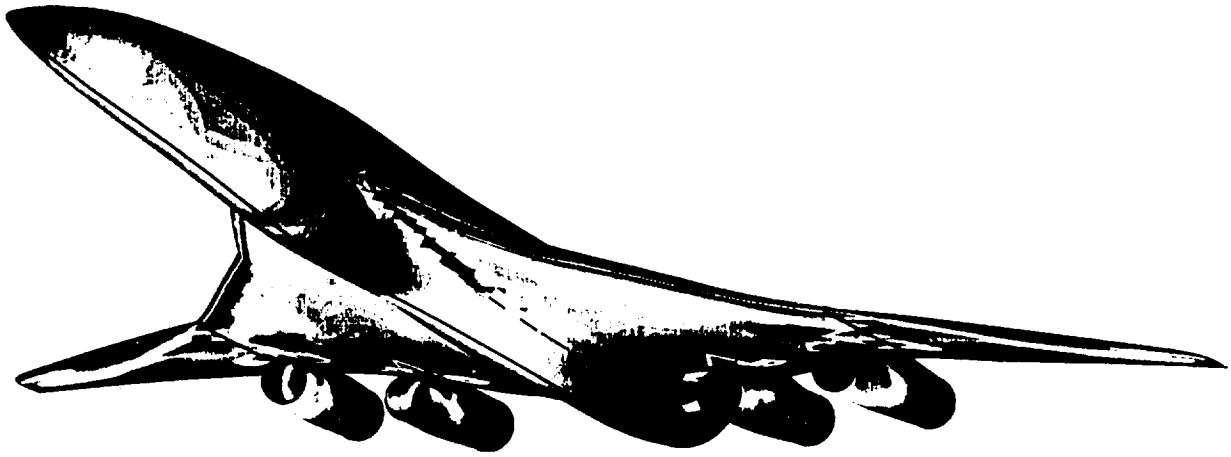


Figure 21. Surface pressure distribution for advanced supersonic transport, Mach 2.4.

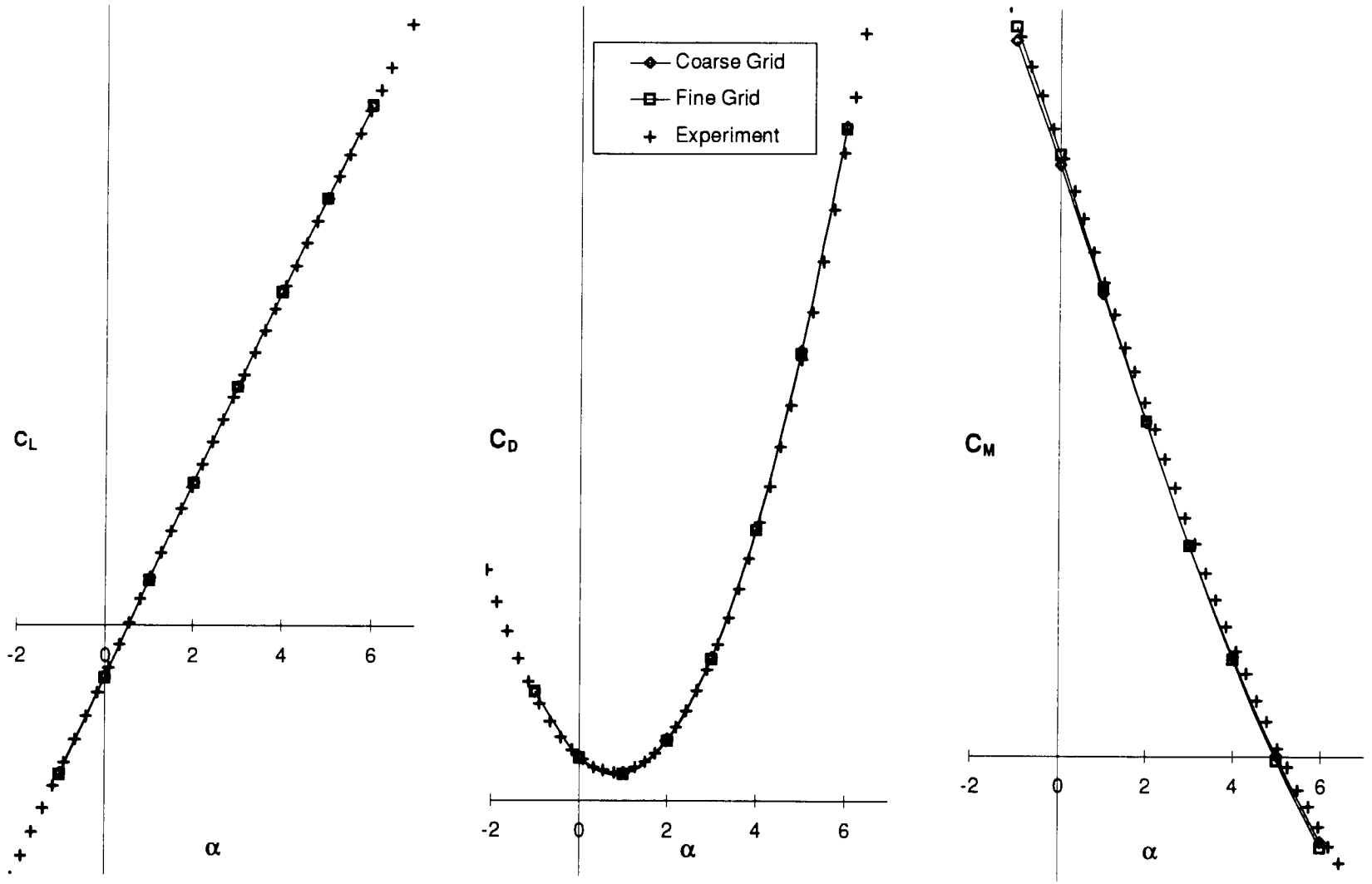



**Figure 22. Lift, drag, and pitching moment for supersonic transport wing-body configuration, Mach 1.8. (CFD data corrected for skin friction drag increment)**

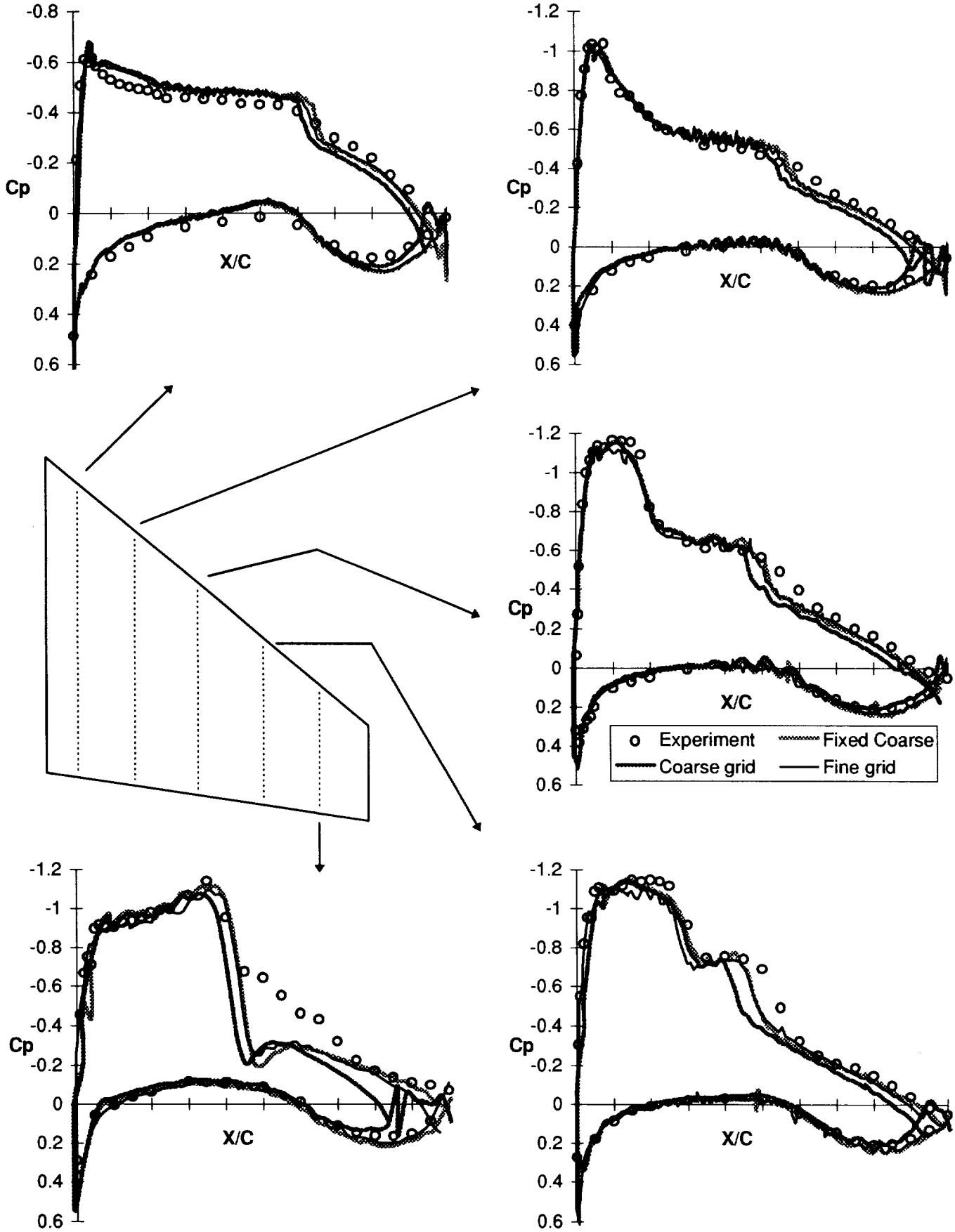
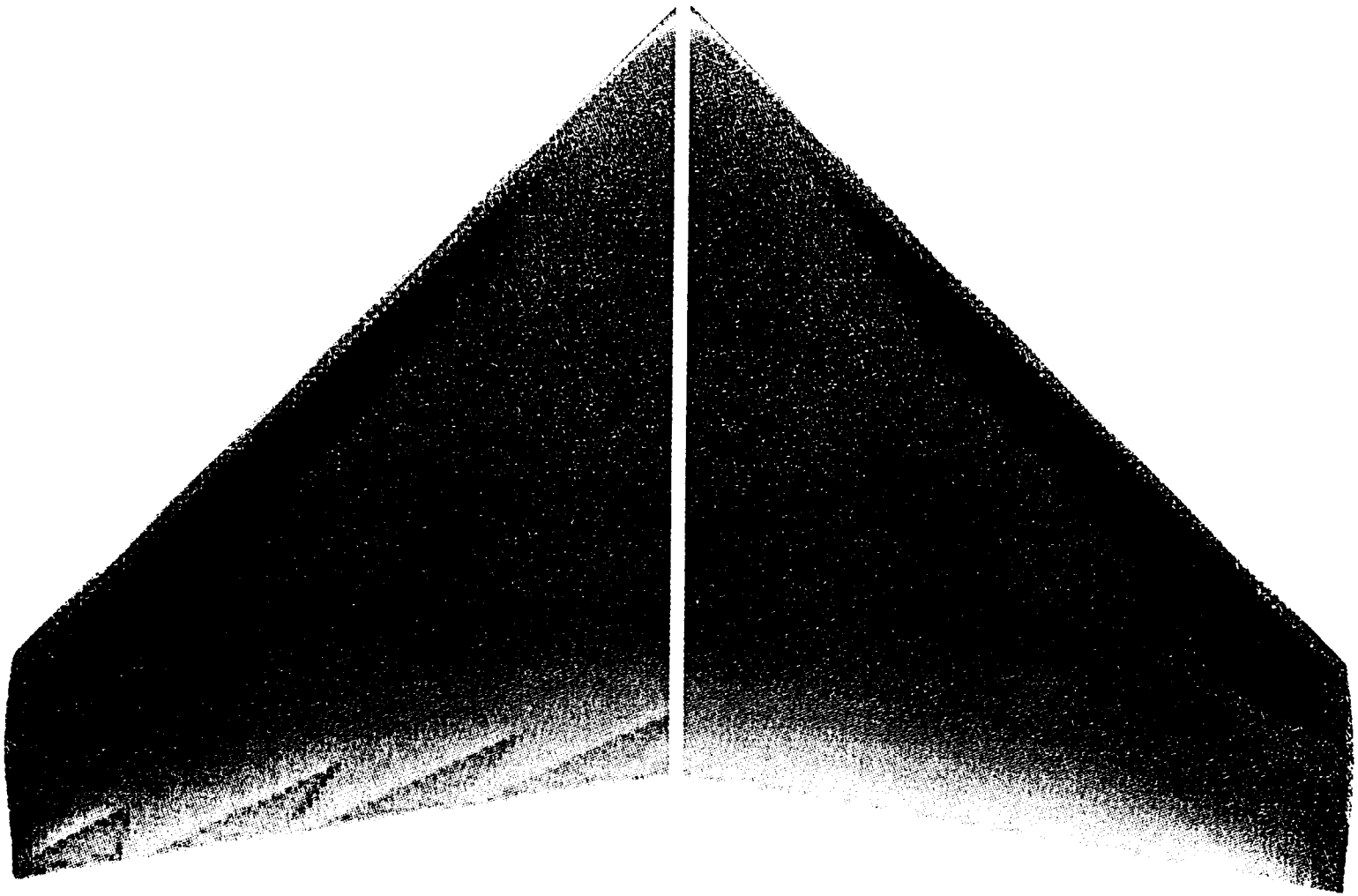



**Figure 23. $C_p$ distributions for Wing C, Mach 0.85, $\alpha = 5°$.**



Original Cartesian Approach

Cartesian Approach with Corrected Multiple Region Cells

**Figure 24. Effect of correcting multiple-region cells on upper surface $C_p$ distributions for Wing C, Mach 0.85, $\alpha = 5°$.**

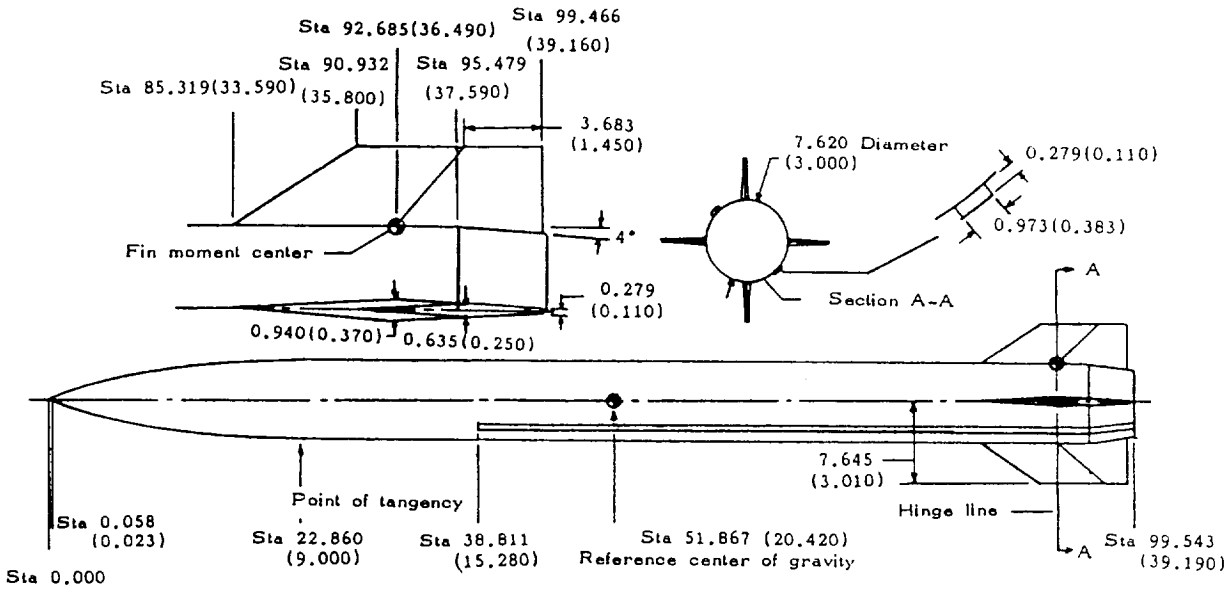



Figure 25. Wingless missile configuration (from Ref. 2). Dimensions are in centimeters (inches).

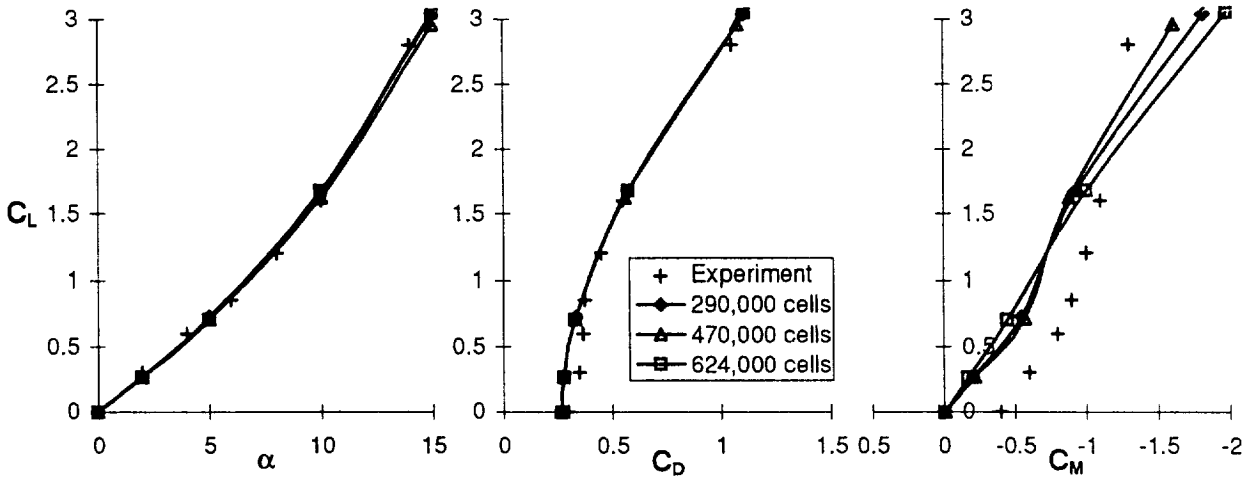



Figure 26. Lift, drag, and pitching moment coefficients for supersonic missile, Mach 2.3.

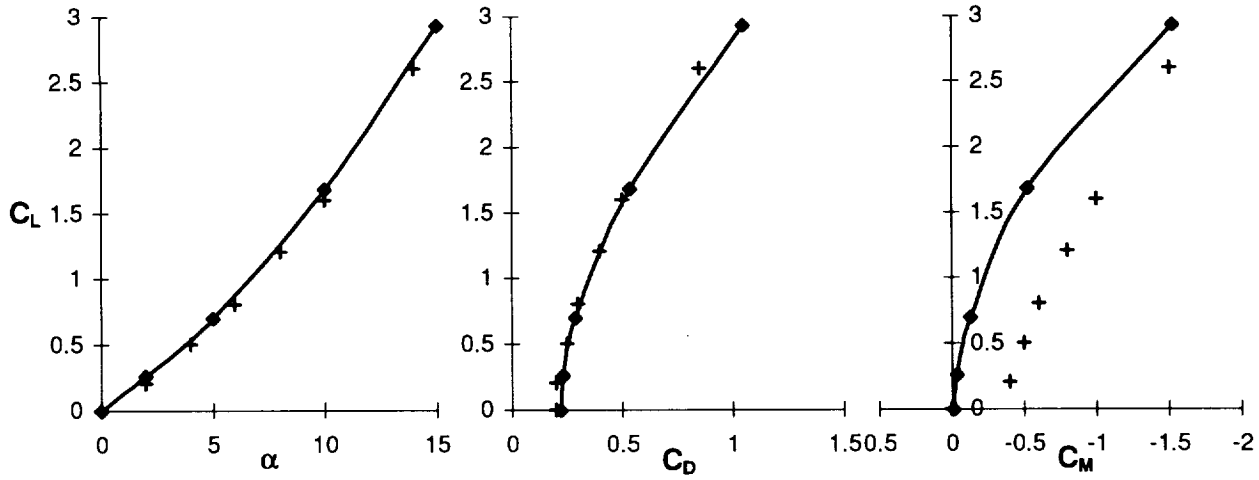



Figure 27. Lift, drag, and pitching moment coefficients for supersonic missile, Mach 2.96.

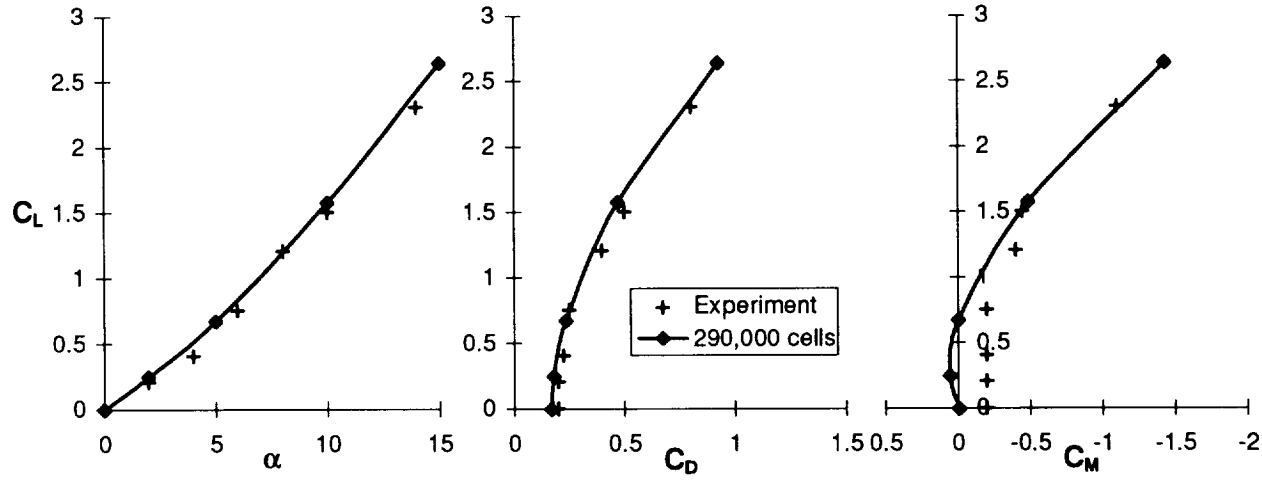



Figure 28. Lift, drag, and pitching moment coefficients for supersonic missile, Mach 3.95.

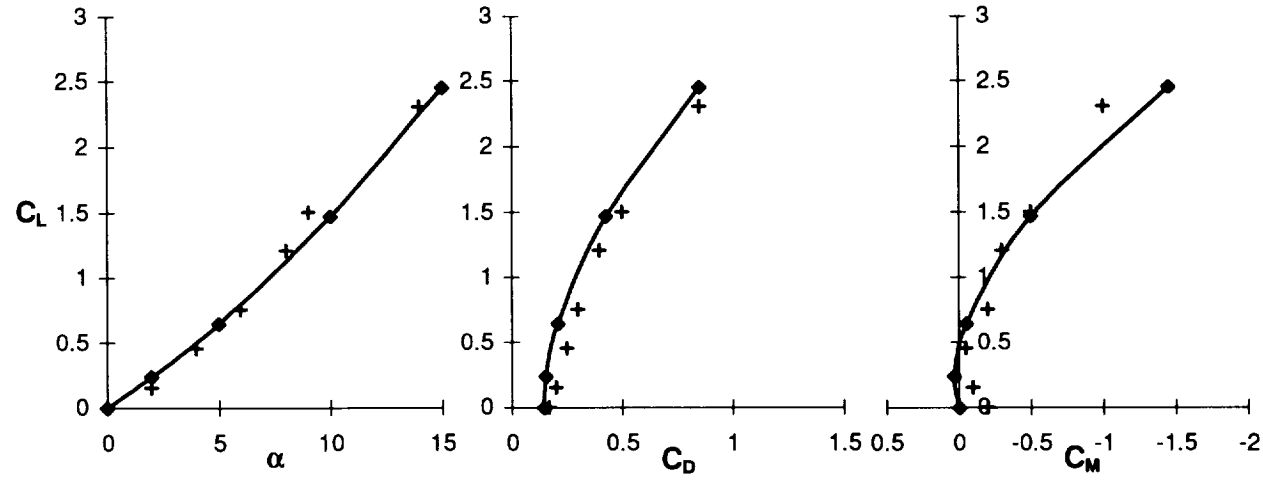



Figure 29. Lift, drag, and pitching moment coefficients for supersonic missile, Mach 4.63.

19951222320

N95-28741

# UNSTRUCTURED CARTESIAN/PRISMATIC GRID GENERATION FOR COMPLEX GEOMETRIES

Steve L. Karman Jr.*
Lockheed Fort Worth Company
Fort Worth, Texas 76101

## SUMMARY

The generation of a hybrid grid system for discretizing complex three dimensional (3D) geometries is described. The primary grid system is an unstructured Cartesian grid automatically generated using recursive cell subdivision. This grid system is sufficient for computing Euler solutions about extremely complex 3D geometries. A secondary grid system, using triangular-prismatic elements, may be added for resolving the boundary layer region of viscous flows near surfaces of solid bodies. This paper describes the grid generation processes used to generate each grid type. Several example grids are shown, demonstrating the ability of the method to discretize complex geometries, with very little pre-processing required by the user.

## INTRODUCTION

The CFD analysis of complex geometries is becoming more prevalent in the early phases of advanced tactical aircraft development. Unstructured grid methods are gaining popularity with the design engineers because of the reduced amount of pre-processing effort required by the CFD practitioner[1,2,3]. Cartesian unstructured methods have recently been developed in which essentially all the grid is automatically generated[4,5,7]. The flowfield around three-dimensional configurations as complex as a fully loaded fighter aircraft have been analyzed using these new Cartesian unstructured methods[10,11]. These analyses generally assume inviscid flow and capture extremely detailed features in the flowfield by using solution adaptive grid refinement. Viscous analyses, on the other hand, are prohibitive due to the large number of Cartesian cells required to resolve viscous regions, such as boundary layers.

Prismatic grid generation methods can generate meshes clustered near body surfaces[6]. Typically, these grids are generated using a method which marches the grid away from the surface in the normal direction. The resulting prismatic grid interfaces with another type of grid used to discretize the global computational domain or is converted to an unstructured grid format and combined with the external unstructured grid[8].

This paper will describe the hybrid grid approach of combining the automatic grid generation versatility of a Cartesian mesh with the efficient clustering capability of a prismatic grid. The combination enables the flow solver to compute solutions about complex geometries without the limitation of assuming inviscid flow. The methods used to generate each part of the hybrid mesh will be described in this paper. The interfacing strategy, used by the flow solver to connect the inner and outer CFD solutions, will also be discussed. Example grids will be shown which demonstrate the use of the techniques on realistic fighter aircraft configurations. No flowfield solutions are presented in this paper, although flowfield features are discernible in the final adapted grids.

*Engineering Group Specialist

Copyright © 1995 by Lockheed Fort Worth Company. All rights reserved.
Published by the National Aeronautics & Space Administration with permission.

PRECEDING PAGE BLANK NOT FILMED

## SYMBOLS

| | |
|---|---|
| A, a, b, c | grid adaption parameters |
| $A_l$, $A_u$ | adaption function lower and upper thresholds |
| $a_x$, $a_y$, $a_z$ | cell face areas in x-, y-, and z-directions |
| $\hat{n}$ | unit normal at grid node |
| $\hat{n}_f$ | unit normal at facet |
| Vol | cell volume |
| x, y, z | Cartesian coordinates |
| $\Delta x$, $\Delta z$, $\Delta z$ | grid spacings in x-, y-, and z-directions |
| $\alpha$ | included angle |
| $\theta$ | angle between node normal and facet normal |
| $\phi$ | normal smoothing parameter |
| $\psi$ | normal influence coefficient |
| $\omega$ | relaxation parameter |
| Subscripts | |
| 1 | minimum x-, y-, or z-coordinate of Cartesian cell |
| 2 | maximum x-, y-, or z-coordinate of Cartesian cell |
| c | cell centroid index |
| f | facet index |
| i | iteration counter |
| j | neighboring node index |

## SURFACE REPRESENTATION

Surface geometry is input to the Cartesian grid method and the prismatic grid method as a triangulated surface mesh. The surface mesh is typically provided by the engineering computer aided design (CAD) package used to define the configuration. By interfacing with the CAD package used by the designers directly, the time required to convert between the designers drawings and CFD surface modeling is virtually eliminated.

The surface in the CAD file is defined as a list of X, Y and Z coordinates and a connectivity in the form of three node numbers corresponding to the global indices of the forming points of each triangle. Geometry facets are oriented such that the surface normals point into the computational domain. Subsets of the facets are grouped together and are associated with a common boundary condition type, such as symmetry, characteristic far-field, surface tangent flow, etc.

The surface triangulation is assumed to be an accurate representation of the surface shape, as opposed to representing the surface using splines or surface polynomials. Therefore, smaller triangles are required to resolve high curvature regions of the geometry. The surface triangles also control the size of the local Cartesian cells in the initial grid generation process.

## CARTESIAN GRID GENERATION

The Cartesian grid is generated using recursive cell subdivision. Cell subdivision is initially based on the user supplied geometry. As the CFD solution evolves, cell subdivision is based on gradients of selected adaption functions. The major elements of the Cartesian grid generation process are described in this section.

### Octree Data Structure

An octree data structure is used to store information for each Cartesian cell during the recursive grid generation process. A subdivided cell produces eight new offspring cells, as shown in Figure 1. The parent cell is retained in the grid after the subdivision. The information stored for each cell consists of the global index of the parent cell, the global indices of the eight children that may exist and the grid level of the cell. The position of each offspring cell, in relation to the parent, is predetermined in the subdivision process. With this information, the neighboring cell indices can quickly be determined. In addition, many of the search procedures are made efficient using the octree data structure. For instance, the cell cutting process, described later, makes extensive use of the parent-child information to quickly traverse the data tree and determine which cells each geometry facet cuts.

### Initial Grid Refinement Based on Geometry

The initial Cartesian grid is generated based on the resolution of the surface triangulation. The process begins with the generation of a cube-shaped root cell that encompasses the entire computational domain. The root cell is at grid level 1 and is subdivided in the X, Y and Z directions, resulting in eight offspring cells at grid level 2, as shown in Figure 1. The process continues with each offspring cell being recursively subdivided according to a length scale criterion.

In the length scale criterion, if the Cartesian cell length scale is larger than the length scale of any facet contained within the cell or touched by the cell, the Cartesian cell is subdivided. A Cartesian cell's length scale is defined as the length of one side of the cell. A geometry facet's length scale can be defined in a number of ways. The most common method is the average length of the three sides of the facet. Other possible definitions include the minimum length of the sides of the facet, the average length of the medians of the facet or the minimum length of the medians of the facet. The user can supply a scale factor associated with each body of the geometry that is used to multiply the facet length scale.

The initial cell subdivision process continues down each branch of the octree data structure until all cells without offspring satisfy the length scale criterion. Recursively, the new offspring cells are tested using the same criterion. The resulting grid contains cells near the boundaries that are proportional in size to the geometry facets near the cell, see Figure 2.

### Grid Smoothing

During the subdivision process several grid quality constraints are enforced. The first constraint limits the number of neighbors a cell can have to no more than four neighbors on any side. This corresponds to limiting the difference in the grid levels between neighbors to one. This constraint is enforced so the octree data structure can be used to rapidly determine the neighbor information of the cells on all grid levels and to simplify the flux calculation performed by the flow solver at each face. Any refinement resulting from this constraint quickly propagates through the grid. The resulting grid varies smoothly from fine resolution cells near the bodies or high gradient regions of the flowfield to coarse resolution cells in the far field.

The second constraint improves the quality of the grid, and ultimately the numerical solution on the grid, by prohibiting rapid changes in grid levels in the mesh. Any cell with finer mesh on opposite sides in any direction is refined, as shown in Figure 3. This constraint eliminates the tendency to

generate coarse-fine-coarse grid regions in the mesh. Numerical accuracy in the flow solver is directly related to the local grid size and is adversely affected by this type of rapid change in mesh size.

## Boundary Cell Cutting

Boundary conditions in the flow solver are imposed on the actual surface shape, as opposed to the stair-stepped surface resulting from the collection of Cartesian cells touching the geometry. Cells at the boundary must therefore be clipped to conform to the surface shape. This cutting process is a critical part of the Cartesian grid generation process. It must be capable of handling totally arbitrary geometries. The cutting process must also be fast, since it is performed after each grid refinement and derefinement process.

The cutting process consists of generating a list of triangular boundary facets resulting from the intersection of the geometry facets with the Cartesian cells. A brute-force approach to generating this boundary facet list would be to test each Cartesian cell for possible intersections with each geometry facet. This approach is extremely slow. A faster approach is to use the octree data structure to test each geometry facet for possible intersections with Cartesian cells. Only cells down the branches of the octree data structure in the vicinity of the geometry facet need be tested. The search procedure consists of testing the limits of a bounding box surrounding a geometry facet for possible intersections with each child of the root cell. If the bounding box intersects with or is contained by the child cell, the offspring of the child cell are then tested for possible intersections with the bounding box. The process continues, recursively, down the branch of the tree to the finest level. Cells at the finest level are then checked for true intersections with the geometry facet.

Once it has been determined that a Cartesian cell without offspring intersects a geometry facet, a list of intersection points is generated. Vertices of the geometry facet contained within the cell are also stored. The intersection points are ordered such that traversing the list of points generates a right-hand rule normal pointing into the computational domain. A centroid of the intersection points is determine by simple averaging of the collected points. The centroid will lie on the geometry facet because the facet is planar and only one facet at a time is processed in this manner. Smaller triangular boundary facets are then generated from the list of ordered intersection points and the computed centroid, see Figure 4. These boundary facets comprise a sub-region of the original geometry facet.

Each Cartesian cell may intersect an arbitrary number of geometry facets. Therefore. the resulting list of boundary facets is generally much larger than the list of geometry facets. A typical ratio of boundary facets to geometry facets for a complex configuration is on the order of 20 or greater.

The actual boundary conditions are applied on these boundary facets. Each boundary facet is associated with only one geometry facet and only one Cartesian cell. Since each geometry facet is a particular type of boundary condition, each Cartesian cell may also contain an arbitrary number of boundary condition types.

## Grid Validity

The Cartesian grid generation process may result in cells that are divided into multiple distinct volumes, such as the cell near the sharp trailing edge region shown in Figure 5. A distinct volume exists above the surface and another exists below the surface. Both of the volumes reside in one Cartesian cell. Cells divided by the geometry in this way are invalid cells, since storage for only one set of conservative variables exists for each cell. Recursive subdivision of invalid cells can correct the situation; however, detecting invalid cells can be difficult and computationally expensive.

Several methods have been devised for detecting invalid cells. The most general method, and the most expensive method, involves sorting the distinct segments of the boundary facets that exist on the six faces of a boundary cell. If the segments occurring on the faces of the Cartesian cell can be

sorted into one curve, then the cell is a valid cell. If the sorting produces more the one curve, then the cell is an invalid cell. This method involves generating a list of segments for each cell, deleting duplicate segments that occur from common edges of boundary facets, and attempting to sort the remaining segments into one distinct curve. Throughout the process, an accurate grid tolerance must be used when comparing endpoints of the segments. A tolerance too small or too large could easily result in multiple closed curves and an incorrect determination of an invalid cell. Another drawback is that the method cannot distinguish between the valid case and the invalid case shown in Figure 6. This may result in excessive grid refinement. However, it may be argued from a numerical accuracy standpoint, that each of these cases should be refined. Each boundary cell can have an arbitrary number of boundary facets. But at what point is the number of boundary facets excessive? It may be wise to refine the ambiguous cases and improve the quality of the mesh in the process.

A much simpler method to detect invalid cells is to sum the X, Y and Z area components of the boundary facets in each cell. If any of the area components sum to zero when the maximum magnitude of the area components in the same direction is non-zero, then the cell may be an invalid cell. This is less precise than the first method and also cannot distinguish between the valid and invalid cases shown in Figure 6. This method will also not detect an invalid cell when the geometry cuts the cell at an angle to the Cartesian grid.

A third cell validity checking approach is a modification of the previous method. Sum the negative and positive contributions to the area components in each direction. If both negative and positive summations of significant magnitude occur in any of the three directions, then the cell may be an invalid cell. Significant magnitude may be defined as a percentage of the face area of an uncut cell. This approach will not distinguish between the valid and invalid cases in Figure 6, but it will detect invalid cells cause by the geometry cutting the cell at an angle to the Cartesian grid, as in Figure 7. This method will also force grid refinement in regions where there is a change in the orientation of surface normal for smooth geometries, such as the wing leading edge shown in Figure 7.

## Solution Adaptive Refinement & Derefinement

Periodically, the grid resolution may be enhanced to capture pertinent flowfield features and to improve the solution accuracy. Adaption of the grid is based on gradients of user selected functions. There are currently ten adaption functions available to users. A few of the more commonly used adaption functions are velocity magnitude, Mach number, pressure, and helicity. Directional adaption parameters are computed for each cell for each selected function, *f*, as

$$a = \frac{\partial f}{\partial x}(\Delta x)^{1+g},\ b = \frac{\partial f}{\partial y}(\Delta y)^{1+g},\ c = \frac{\partial f}{\partial z}(\Delta z)^{1+g}$$

$$A = \sqrt{a^2 + b^2 + c^2}$$

The variable in the exponent of the length scale multiplier is used to decrease the effect of discontinuities in the flowfield on the grid adaption[9]. Generally, *g* is set to a value of 1.0. A value of 0.0 corresponds to computing the gradients of the function in computational space. Thus, a shock would produce a constant valued adaption function as the grid was refined and could inhibit refinement from taking place in smoother regions of the flow where large cells exist. Values of g greater than 1.0 would lessen the effect of discontinuities even further and have been used successfully in fully supersonic flowfields to force more rapid grid refinement of the entire flowfield.

The magnitude, *A*, of the adaption parameters is statistically averaged. Then, during the derefinement process, a lower threshold value of the adaption function is computed as

$$A_l = m - d\sigma ,$$

where $m$ is the mean, $\sigma$ is the standard deviation and $d$ is a user defined constant, typically set to 1.0. Any cell, without children, with all three directional adaption parameters less than $A_l$ is marked for deletion. If all eight children of a parent cell are marked for deletion, then the children are deleted from the cell list. The parent cell would reclaim the collective volume of the computational domain previously occupied by the children.

During the refinement process, an upper threshold value of the adaption function is computed as

$$A_u = m + e\sigma .$$

The variable $e$ is another user defined constant typically set to 1.0. Any cell, without children, with any one of the directional adaption parameters greater than $A_u$ is marked for refinement.

## Additional Considerations

The refinement process could, conceivably, continue to add cells to the grid indefinitely. Therefore, some limits must be imposed. The first limit is on the minimum and maximum cell size allowed in the grid. No cell can be refined with a length scale less than the minimum cell size specified by the user and no cell can exist without children with a length scale greater than the maximum cell size specified by the user. These cell size limits are usually unnecessary because of the next limiting process imposed on the refinement process.

The user specifies a "target" number of cells desired in the final grid. This target must be slightly less than the maximum dimension of the code in order to allow the code to enforce the smoothness criteria mentioned earlier. The user can also specify a maximum number of new cells added per adaption function during each grid refinement process. Limiting the number of cells added per refinement allows the solution to evolve slowly between refinements. This prohibits the grid from reaching the maximum number of cells prematurely, before the flowfield can adequately develop. Additionally, the user may specify a refinement box which delineates a region in the computational domain where adaptive grid refinement takes place.

During the refinement process the cells are ranked from highest to lowest, according to adaption parameter A. The marking of the cells for refinement proceeds from the top of the list down until the estimated total number of cells exceeds the target maximum number of cells or the estimated number of new cells added exceeds the specified maximum number added per adaption function. Thus, if insufficient deletion took place to reduce the current number of cells below the target maximum, no refinement would take place. In this case, the multiplier on the standard deviation in the lower threshold value could be reduced or the target number of cells could be increased. The effect of these limiting processes is to produce a near optimum grid for the target number of cells and the selected adaption functions.

## Cell Face Areas, Volumes and Centroids

The flow solver for the CFD code that uses this Cartesian mesh requires accurate computation of the face areas and volumes of all cells. The higher order extrapolation procedure also requires the location of the centroid of each face and the centroid of the cell. For uncut cells the face areas, volumes and centroids are easily computed.

$$a_x = \Delta y \Delta z,\ a_y = \Delta x \Delta z,\ a_z = \Delta x \Delta y$$

$$x_c = \frac{(x_1 + x_2)}{2}, y_c = \frac{(y_1 + y_2)}{2}, z_c = \frac{(z_1 + z_2)}{2},$$

$$Vol = \Delta x \Delta y \Delta z$$

For cube shaped cells, the spacing in each direction, $\Delta x$, $\Delta y$, $\Delta z$, are equal. The centroids of the faces are the appropriate components of the cell centroid, ($x_c$, $y_c$, $z_c$).

The boundary facets contain the necessary information for computing the face areas, centroids and volumes of the cut cells. Edges of boundary facets that exist on each face of a cut cell are used to compute the exposed area and centroid of the face. The face areas and the boundary facets are then used to compute the cell volume and cell centroid.

An example of the exposed area computation of a face of a cut Cartesian cell is shown in Figure 8. The shaded area in the top figure represents the portion of the face contained inside the geometry. The unshaded region is the exposed area or the portion of the face existing in the computational domain. An elaborate procedure has been developed to compute the exposed area of the face. The procedure involves summing areas swept out by the edges of the boundary facets that exist on the face.

A summation point is selected, such as the lower left corner of the face shown. The areas swept out by the two vertices of each edge and the summation point are summed. The ordering of the edge points determines whether the computed area is positive or negative. The positive area that results from this process is shown as the initial area of the face.

The two vertices of each edge are then projected to the top and right boundaries of the face. The "anti-areas" swept out by the projected vertices and the summation point are summed. The ordering of the projected points determines whether the computed area is positive or negative. The negative area that results from this process is shown as the anti-area of the face.

The area and anti-area of the face are combined to produce the actual exposed area of the face. In this case the combination produces the negative area shown as the combined area. Adding this negative area to the area of the uncut face produces the area of the exposed region shown as the resulting area. The same procedure is applied on each of the faces of cut cells that are intersected by boundary facets.

The centroids of the faces are computed by summing the area weighted centroids of the swept out regions during the previously described steps and dividing the result by the actual face area. The individual swept out regions are triangles, so the centroids are simply the average of the three forming nodes of the triangle.

Volumes of cut cells are then computed by summing volume of sub-components of the cell. A cell summation point is selected, such as the centroid of the uncut Cartesian cell. Tetrahedra are formed from the summation point and the forming nodes of the boundary facets. The volume of each tetrahedra contributes to the total volume of the cell. The orientation of the facet forming nodes in relation to the summation point determines whether the computed tetrahedral volume is positive or negative. Prismatoids are formed from the summation point and the area of each face of the cell. The volume of each prismatoid is positive and contributes to the total volume of the cell. An example of the sub-components of a cut cell are shown in Figure 9.

Cell centroids are computed from a volume weighted average of the centroids of the sub-components of the cell just described. The centroids of the tetrahedra are simply the average of the four forming nodes. The centroid of the prismatoid is assumed to be the point one fourth the distance from the face centroid to the cell summation point.

## PRISMATIC GRID GENERATION

The prismatic grid is generated by marching a triangulated surface mesh outward. The major elements of the process are the surface triangulation, the computation of the surface normals at each grid point, the marching step size, and the prevention of grid crossing in concave and convex regions of the surface.

### Surface Triangulation

The surface triangulation produced by the CAD program described earlier is used in the development of the prismatic grid. In this case, however, the resolution of the surface mesh is more than a description of the geometry. It is the starting layer of the prismatic grid. Therefore, facet resolution in critical regions other than high curvature regions is important. The prismatic grid is currently not refined by the CFD code during the solution process, so adequate resolution of the surface in critical regions must be supplied by the surface triangulation.

### Surface Normals

The triangular surface facets are marched outward along carefully computed normal vectors from each node point, see Figure 10. The normal vector at each node must be constructed such that it is visible to each facet containing that node[6]. This ensures that the developing grid layers in convex regions do not cross and produce negative prismatic cell volumes.

The initial normal vector at each node is computed as the weighted average of the normals of the common faces

$$\hat{n}^0 = \frac{\sum_{f}^{mf} \hat{n}_f \alpha_f}{\sum_{f}^{mf} \alpha_f},$$

where $\hat{n}_f$ is the unit normal of common faces, and $\alpha_f$ is the included angle of the face as shown in Figure 10. The variable *mf* is the number of faces surrounding the node.

An iterative procedure is then used to improve the normal vector at each node using a linear combination of a weighted average of the normal vectors of the common faces and a weighted average of the position vectors of the neighboring nodes projected to the next layer.

$$\hat{n}^{i+1} = \hat{n}^i + \omega \left( (1-\phi) \left( \frac{\sum_{f}^{mf} \alpha_f \hat{n}_f \psi_f^i}{\sum_{f}^{mf} \alpha_f \psi_f^i} \right) + \phi \left( \frac{\sum_{j}^{nn} \dot{r}_j d_j}{\sum_{j}^{nn} d_j} \right) \right)$$

$$\psi_f^i = 1 + \theta.$$

The weighted average of the face normals is once again based on the included angle of each common face. The common face normal vector, $\hat{n}_f$, is multiplied by $\psi$, which is a function of the angle between the current node normal vector and the face normal vector, $\theta$. The position vector of the neighboring nodes projected to the next level is given by $\dot{r}_j$ and is weighted by the distance to the node, $d_j$. The variable $\phi$ varies from 0.0 to 0.5 and controls the amount of smoothing of the normal vector. With no smoothing, the scheme tends to minimize the angle between the resulting node normal and common

face normals. With smoothing, the grid expands as it grows outward, approaching a mesh where the tangential spacing between the nodes is equidistant. A typical value for the smoothing parameter is 0.25. An under-relaxation factor, $\omega$, is used to update the new normal vectors. The iterative procedure continues until the changes in the normal vectors are negligible or a maximum iteration limit is reached.

The use of the included angle in the averaging procedure eliminates the tendency to get skewed vectors for cases where a node has many small included angle facets on one side. This situation may occur at sharp trailing edges. This normal calculation procedure has been tested on many complex shapes and has produced valid surface vectors.

Additional control is added in extreme convex and concave regions of a geometry. The smoothing portion of the equation is turned off for the nodes where the angle between the face normals common to any given facet edge exceeds a user specified amount, such as 60 degrees. The normals at such nodes are then improved using only the face normal portion of the iterative procedure.

### Marching Step Size

The grid is advanced to the next layer by a specified spacing increment. The spacing increment can be equally spaced, generated from a prescribed normal distribution or based on the minimum radius of an inscribed circle for the facets on each layer. The marching step size at extremely concave nodes is slightly increased. The marching step size at extremely convex nodes is slightly decreased. These adjustments help to improve the quality of the grid at the next layer. The effect of the adjustments is to reduce the maximum angle between face normals on each subsequent grid layer, i.e. smoothing out the waviness.

The total thickness of the prismatic grid in the normal direction is usually just large enough to resolve the estimated boundary layer thickness for viscous analyses or large enough to improve the invalid cell situation at sharp edges for inviscid analyses. At each new layer the normals at each node are recomputed using the above scheme and the process continues until the desired number of prismatic layers is completed. Any portion of the boundary layer in viscous analyses not resolved by the prismatic grid will be resolved by the refined Cartesian grid using the grid adaption scheme.

### Prevention of Grid Crossing

Grid crossing in convex regions of the surface is prevented by carefully computing the surface normals, as described earlier. Grid crossing in concave regions of the surface is controlled by reducing the marching step size. As each layer is marched outward, the current grid layer is checked for grid crossing. When grid crossing is detected, the local marching step size is reduced. The marching step sizes across the current layer are then smoothed using a Laplacian type smoothing. The nodes are projected outward once again and rechecked for grid crossing. This process continues until the crossing is eliminated.

## CARTESIAN/PRISMATIC GRID INTERFACE

The Cartesian grid treats the outer layer of the prismatic grid as another boundary. The Cartesian grid generation, cell cutting and so forth are performed in the usual manner, since the outer layer of the prismatic grid is made up of triangular elements like all other defined geometries. During the solution process, the prismatic grid and the Cartesian grid interact through the flux calculations at the interface boundary. Fluxes are computed at the interface boundary facets using data from the interior of the prismatic grid and the local Cartesian grid cells. Fluxes are computed at the interface boundary using the same upwind or central differencing schemes as used with the Cartesian grid faces and the interior prismatic grid faces. The computed fluxes are distributed to each side of the interface in a fully conservative manner.

## EXAMPLE GRIDS

Several example cases are included to demonstrate the capability of the grid generation procedures to discretize complex geometries and complex flowfields. Some of the cases use only the Cartesian grid, while one case uses the hybrid grid system. The final case includes only the prismatic grid for a complex store configuration. Flowfield solutions were obtained for all of the Cartesian grid-only examples and the single hybrid grid example. Grid refinement of the Cartesian grid was used to resolve pertinent flowfield features in these cases. Since this paper pertains to the grid generation aspect of the problems, no flowfield solution data will be shown.

### F16 Forebody

An inviscid supersonic flowfield for a F16 forebody/inlet geometry was computed using the Cartesian grid system. The surface model included extensive geometric detail, including the inlet duct back to the compressor face, the diverter section between the inlet and the underside of the fuselage, and the CD band antenna on the underside of the nose. Grid refinement was based on two adaption functions, Mach number and total pressure. The purpose of the analysis was to gain an understanding of the shock structure and total pressure field entering the inlet. Qualitative results from this analysis helped guide engineers in grid generation for a structured grid Navier-Stokes solution. A symmetry plane cut through the mesh is shown in Figure 11. Adaption to Mach number is evident in the increased grid resolution at the shocks, while the adaption to total pressure is responsible for the refinement aft of the CD band antenna. The final grid contained 601,513 Cartesian cells.

### F16 With Various Store Loadings

Several solutions were computed for a full F16 configuration with various types of weapons loadings. The F16 aircraft geometry included accurate modelling of the inlet duct back to the compressor face, the nozzle duct forward to the turbine face, the ventral fin, the wing with tip missile rail, and the horizontal and vertical tails. The Cartesian grids for three different weapons loadings are shown in Figure 12, Figure 13, and Figure 14. An inviscid transonic flow was computed in each case and the adaption functions were velocity magnitude and static pressure.

The weapons shown in Figure 12 include a 600 gallon fuel tank and 3 CBU58s mounted on a triple ejector rack. The constant-X cutting plane located aft of the wing trailing edge reveals the improved resolution of the three-dimensional shock structure on the wing upper surface. The final grid contained 953,385 Cartesian cells.

Figure 13 shows a AIM 9 missile along with the fuel tank and the CBU58s. The constant-X cutting plane is located at the mid-point of the wing leading edge and shows the increased resolution of the region about the CBU58s. A total of 865,993 Cartesian cell are contained in the final grid.

The additional components shown in Figure 14 include the AIM 9, a 370 gallon fuel tank, an ALQ119 ecm pod and a MK84 with pylon. Increased grid resolution about the MK84 store is evident in the figure. This analysis was part of a quasi-steady trajectory analysis of the MK84 store separation. Steady-state solutions were computed for several instances along the trajectory path. The computed forces and moments were used, in conjunction with a six degree of freedom package, to compute the new store locations. The process is analogous to the wind tunnel testing technique known as Captive Trajectory Simulation.

### Wing/Pylon/Store

A hybrid grid was used to discretize the computational domain surrounding a wing/pylon/store configuration. The geometry was a clipped 45 degree swept delta wing with a pylon located at mid-span and a generic store[10,11,12]. This analysis used a prismatic grid about the store combined with a

Cartesian grid for the wing, pylon and the remainder of the domain. The extent of the prismatic grid was limited because of the close proximity of the store to the base of the pylon. The prismatic grid consisted of 8,838 cells per layer in 5 cell layers for a total of 44,190 cells. The final Cartesian grid consisted of 307,361 cells in 17 grid layers. Periodic grid refinement on the Cartesian grid was performed based on the gradients of Mach number and pressure. A side view of the Cartesian grid is shown in Figure 15. A refinement box that extended just above the wing geometry was used to limit the Cartesian grid refinement to the region in the vicinity of the store. The prismatic grid about the store is shown in Figure 16. Less than half of the gap between the store and the pylon is discretized by the prismatic grid. The remaining gap was discretized with the Cartesian grid. The use of the prismatic grid about the store resulted in approximately half as many Cartesian cells as required for a Cartesian-only solution.

### MK84 Store

A prismatic grid was generated for an isolated MK84 store geometry, see Figure 17. The MK84 geometry is very similar to the generic store shown in the previous case, e.g. a cylindrical body of revolution with four tail fins. A five layer prismatic grid was generated for demonstration purposes only. No solution has been obtained for this geometry. The thickness of the layers was restricted to approximately 0.5 inches, due to the 90 degree corners at the fin/body juncture. The grid was equally spaced in the direction normal to the surface. Grid smoothing is necessary to ensure the prismatic elements march out of the corner regions properly. The marching step size for each layer also has to be limited to prevent grid crossing in the corners. A thicker prismatic grid is possible with additional layers, but would make viewing the grid more difficult than it currently is. Had this been an actual prismatic grid for a viscous analysis, the number of layers would be larger, and the normal grid spacing would be clustered toward the surface, as can be seen in the 21 layer grid shown in Figure 18 and Figure 19.

## CONCLUSIONS

The methods for generating unstructured Cartesian meshes and triangular-element prismatic meshes have been described. Example grids for various configurations have been shown for Cartesian-only grids, hybrid grids, and prismatic-only grids. Grid adaption for the Cartesian grid has been demonstrated in several of the examples where a flowfield solution was generated using the CFD solver developed by the author. Highly three-dimensional flowfield features are apparent in the meshes, as the refinement scheme detected gradients in the selected adaption functions. The flexibility of the prismatic grid was demonstrated in an actual hybrid grid solution for a wing/pylon/store geometry where a prismatic grid was employed around the store. The resulting inviscid solution made more efficient use of fewer Cartesian cells to resolve the remainder of the domain. A prismatic grid suitable for viscous solutions was shown for a complex store geometry. The ability to control the thickness and normal distribution of the prismatic layers was demonstrated.

## REFERENCES


1. Frink, N. T., Parikh, P., and Pirzadeh, S., "A Fast Upwind Solver for the Euler Equations on Three-Dimensional Unstructured Meshes," AIAA-91-0102.

2. Potsdam, M. A., Intemann, G. A., Frink, N. T., Pirzadeh, S., "Wing/Pylon Fillet Design Using Unstructured Mesh Euler Solvers," AIAA-93-3500.

3. Pirzadeh, S., "Viscous Unstructured Three-Dimensional Grids by the Advancing-Layers Method," AIAA-94-0417.

4. De Zeeuw, D., Powell, K. G., "An Adaptively-Refined Cartesian Mesh Solver for the Euler Equations," AIAA-91-1542-CP.

5. Melton, J. E., Enomoto, F. Y., Berger, M. J., "3D Automatic Cartesian Grid Generation for Euler Flows," AIAA-93-3386-CP.

6. Kallinderis, Y., Ward, S., "Prismatic Grid Generation with an Efficient Algebraic Method for Aircraft Configurations," AIAA-92-2721-CP.

7. Melton, J. E., Pandya, S. A., Steger, J. L., "3D Euler Flow Solutions using Unstructured Cartesian and Prismatic Grids," AIAA-93-0331.

8. Ward, S., Kallinderis, Y., "Hybrid Prismatic/Tetrahedral Grid Generation for Complex 3-D Geometries," AIAA-93-0669.

9. Warren, G. P., Anderson, W. K., Thomas, J. L., Krist, S. L., "Grid Convergence for Adaptive Methods," AIAA-91-1592.

10. Karman, S. L. Jr., "SPLITFLOW: A 3D Unstructured Cartesian/Prismatic Grid CFD Code for Complex Geometries," AIAA-95-0343.

11. Welterlen, T. J., Karman, S. L. Jr., "Rapid Assessment of F-16 Store Trajectories Using Unstructured CFD," AIAA-95-0354.

12. Parikh, P., Pirzadeh, S., Frink, N. T., "Unstructured Grid Solutions to a Wing/Pylon/Store Configuration Using VGRID3D/USM3D", AIAA-92-4572.

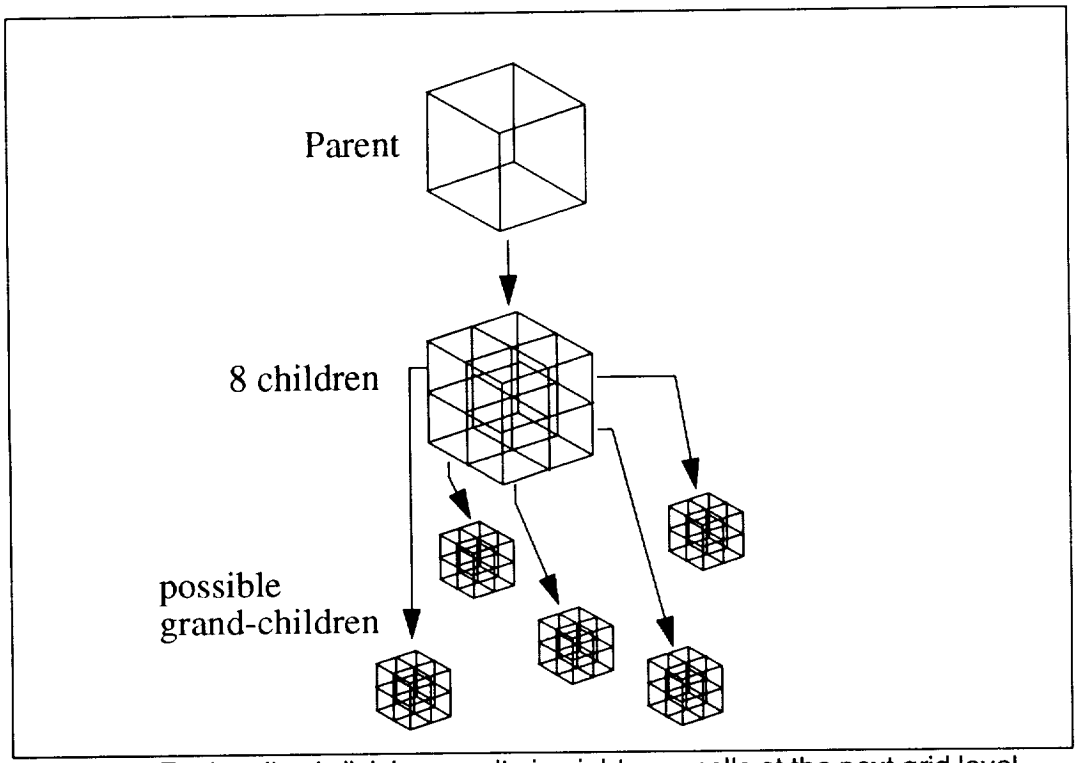



Figure 1. Each cell subdivision results in eight new cells at the next grid level using an octree data structure.

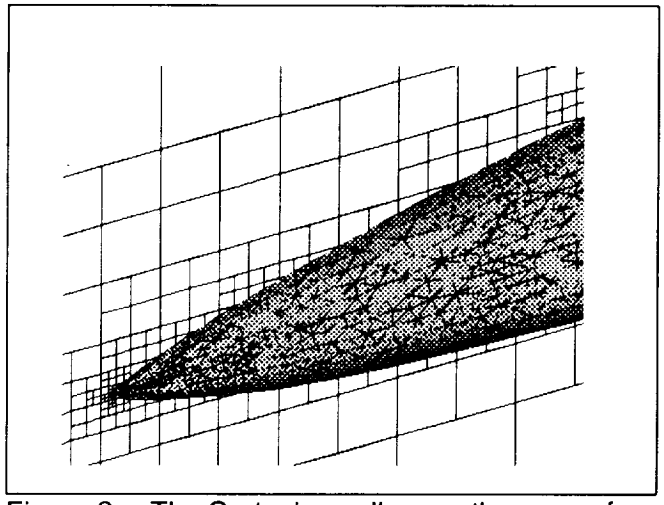


Figure 2. The Cartesian cells near the nose of a fighter configuration are proportional in size to the local geometry facets.

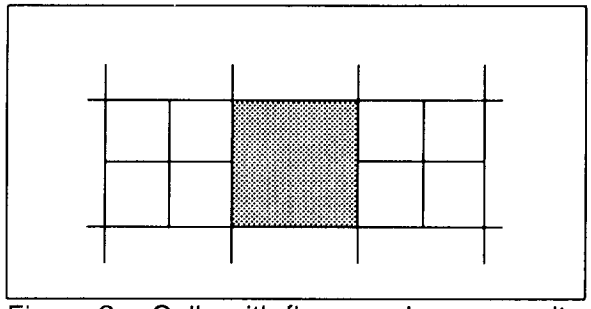


Figure 3. Cells with finer mesh on opposite sides are refined.

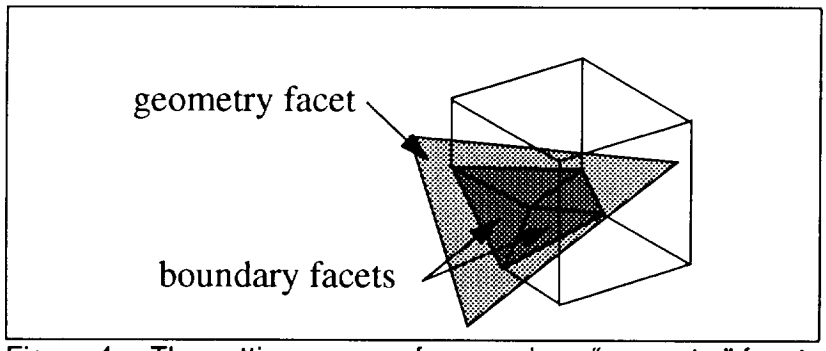



Figure 4. The cutting process for any given "geometry" facet may produce multiple "boundary" facets.

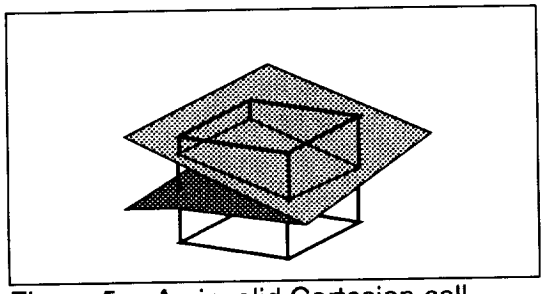


Figure 5. An invalid Cartesian cell caused by the cutting of a sharp trailing edge.

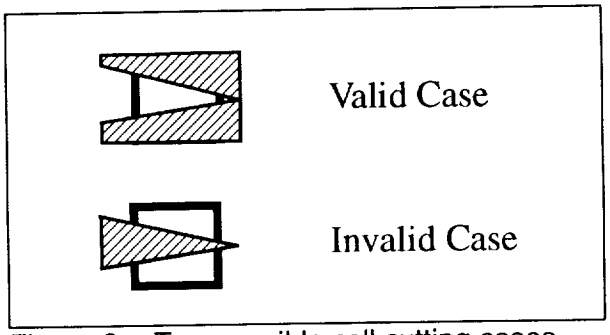



Figure 6. Two possible cell cutting cases.

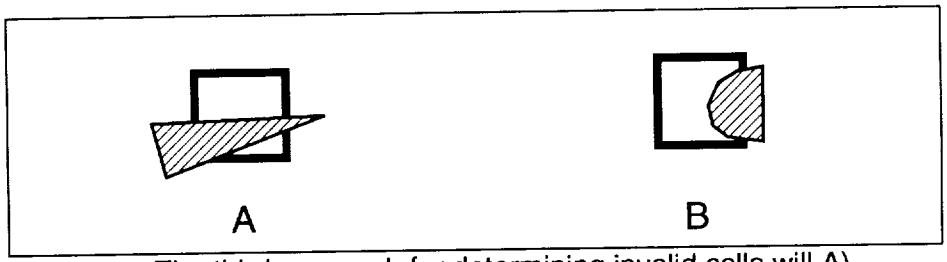



Figure 7. The third approach for determining invalid cells will A) correctly detect the cutting of cells at an angle to the Cartesian grid and B) detect changes in surface orientation.

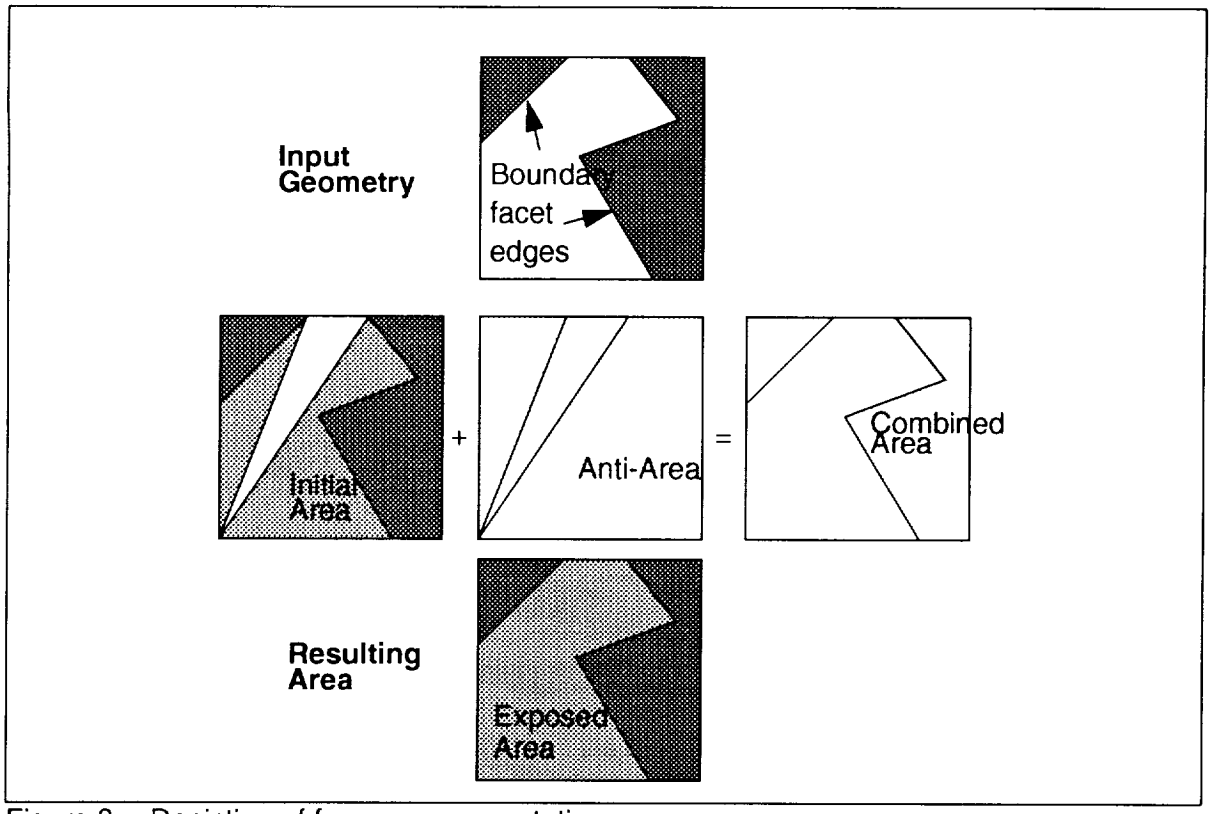



Figure 8. Depiction of face area computation process.

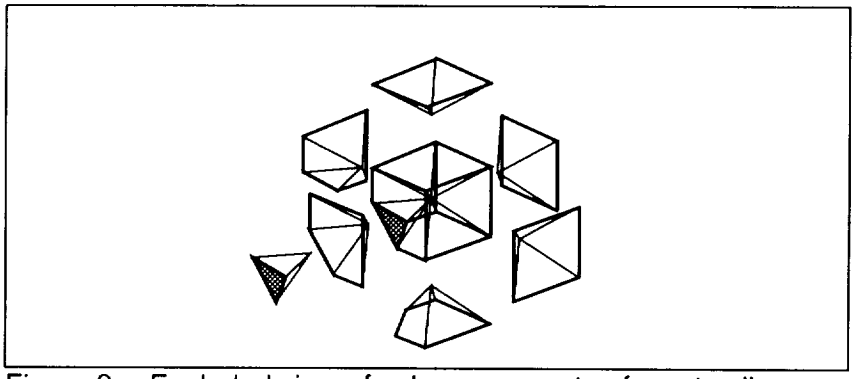


Figure 9. Exploded view of sub-components of a cut cell containing a single boundary facet.

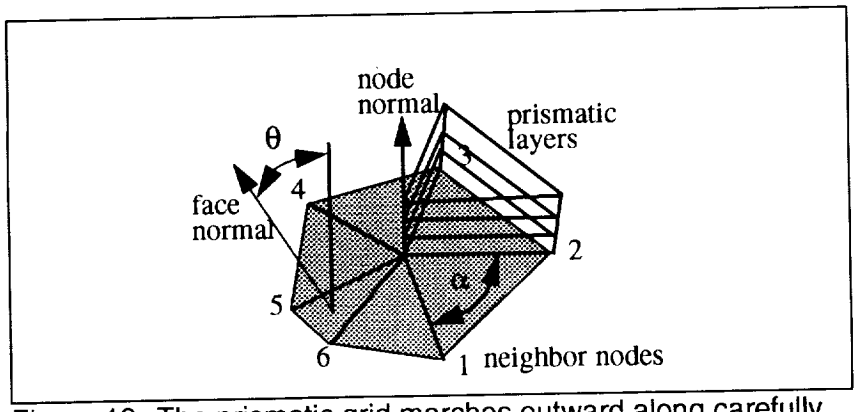


Figure 10. The prismatic grid marches outward along carefully computed normal vectors.

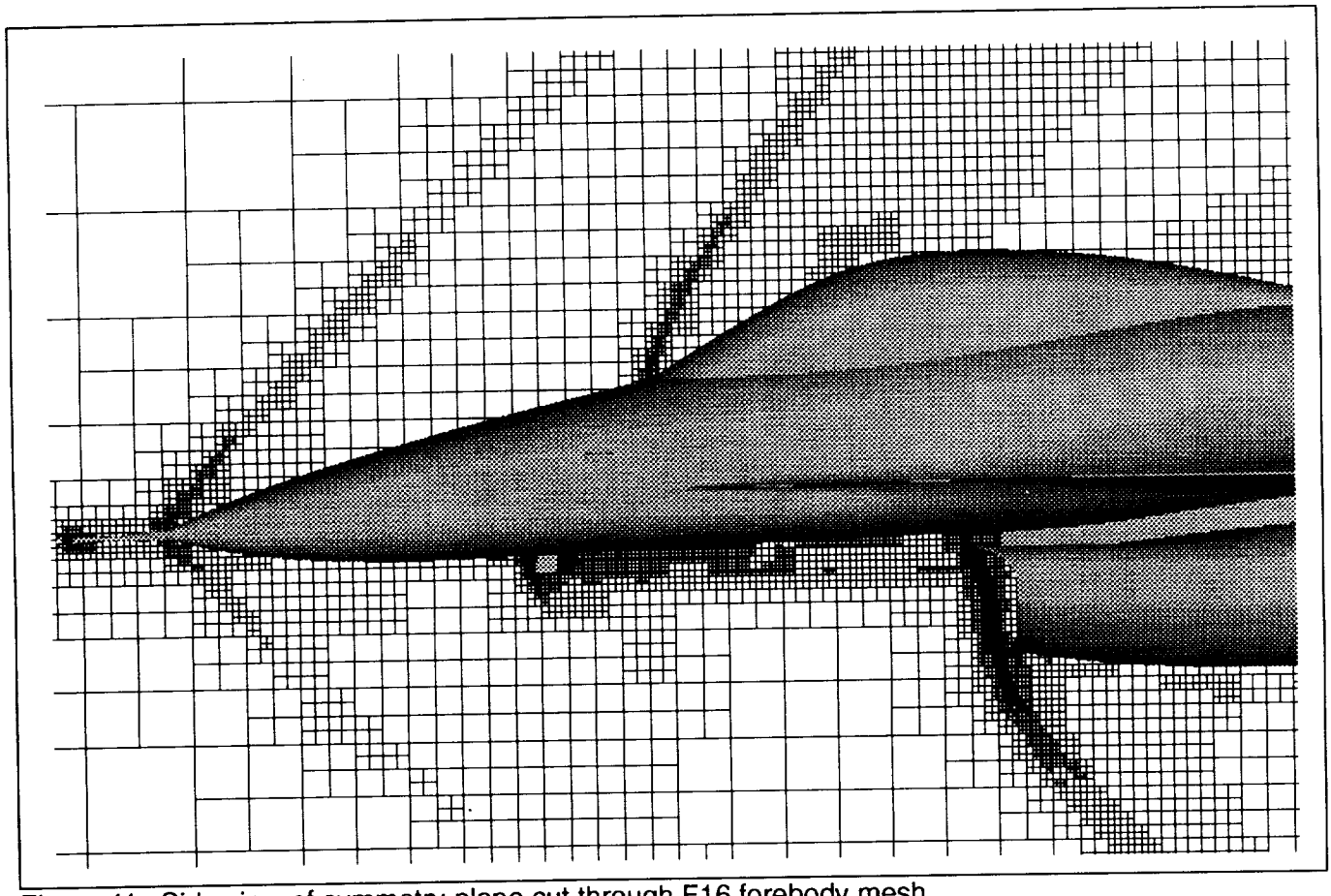


Figure 11. Side view of symmetry plane cut through F16 forebody mesh.

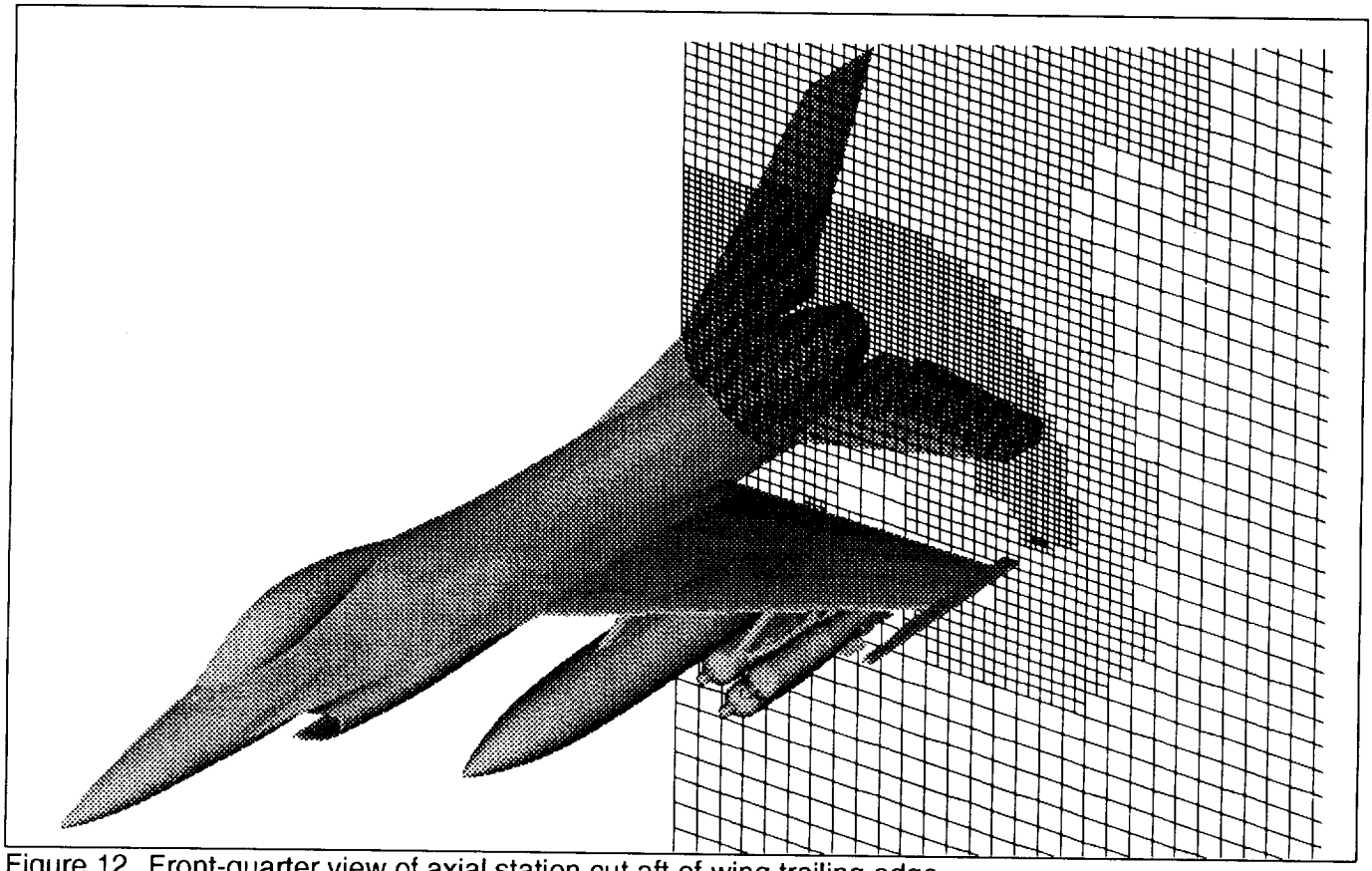


Figure 12. Front-quarter view of axial station cut aft of wing trailing edge.

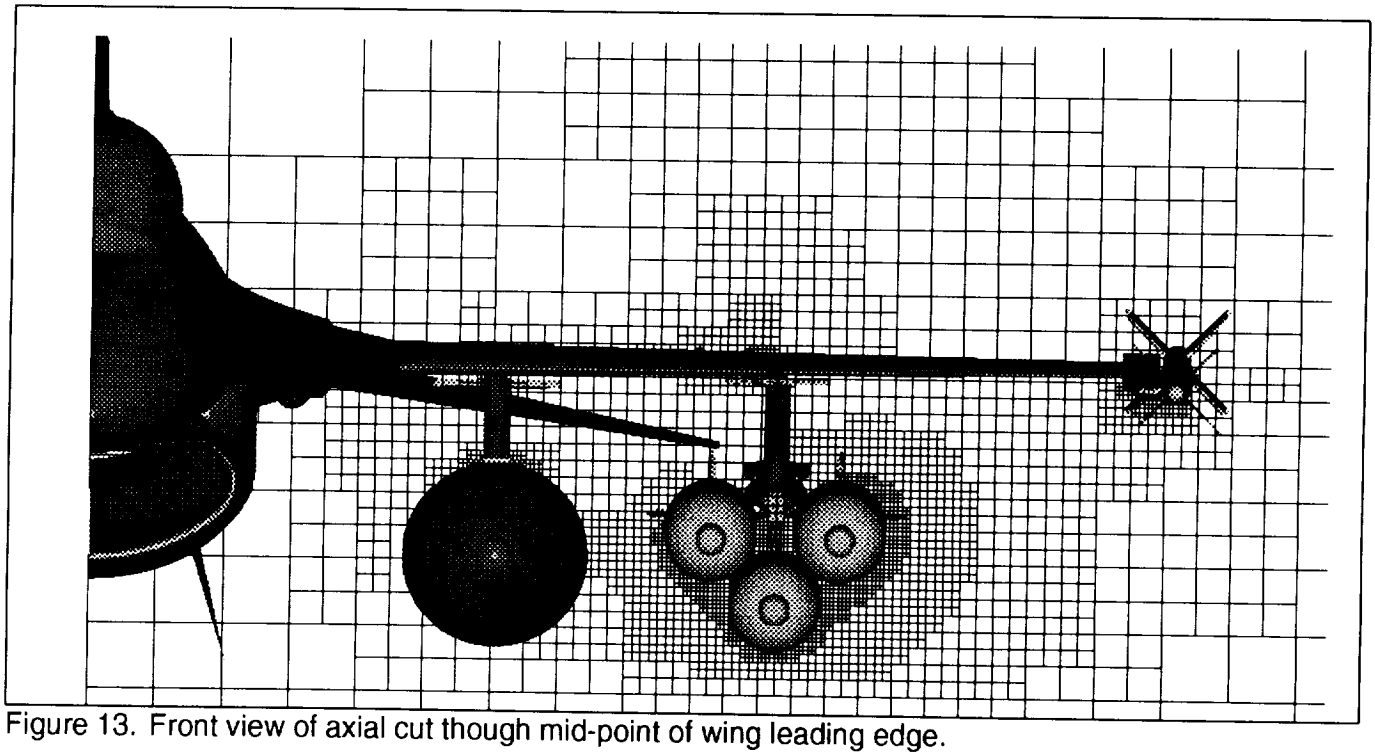


Figure 13. Front view of axial cut though mid-point of wing leading edge.

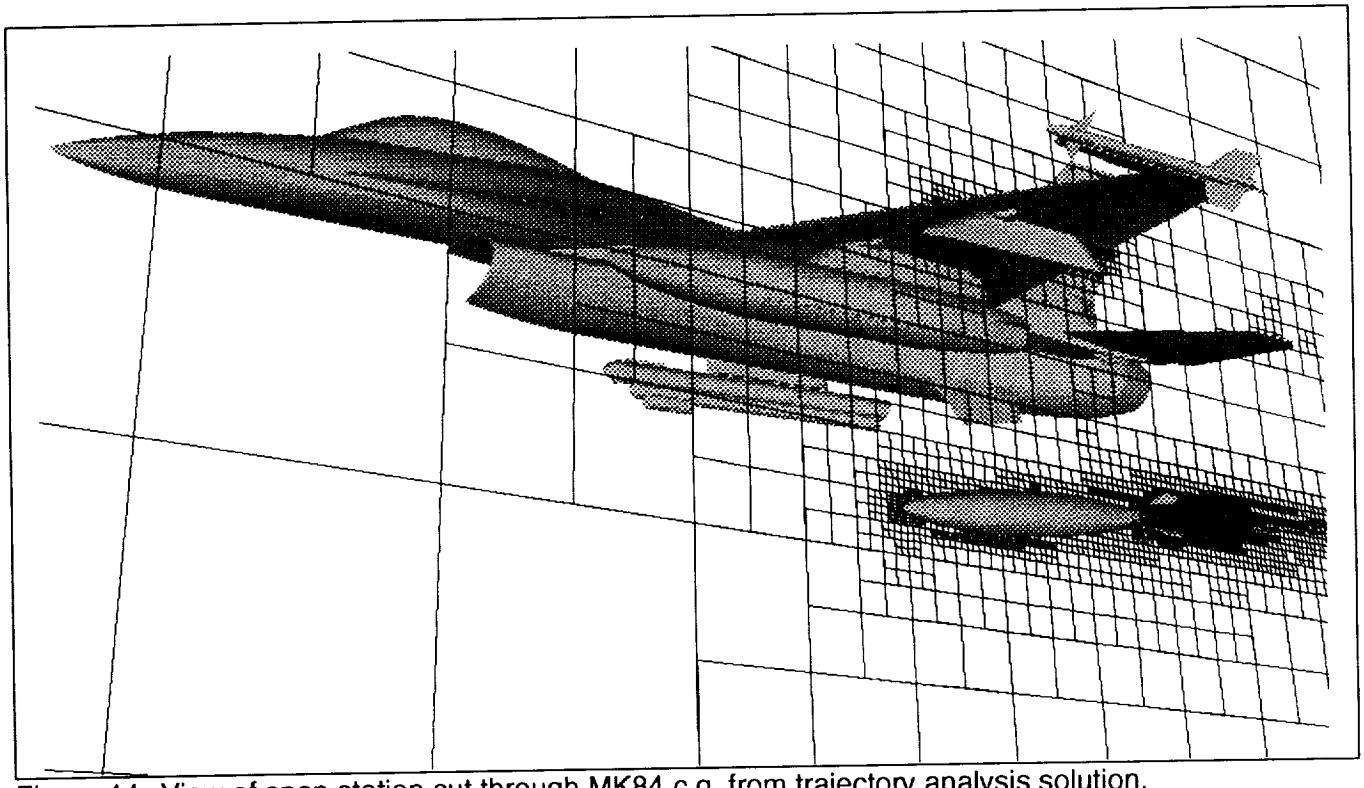


Figure 14. View of span station cut through MK84 c.g. from trajectory analysis solution.

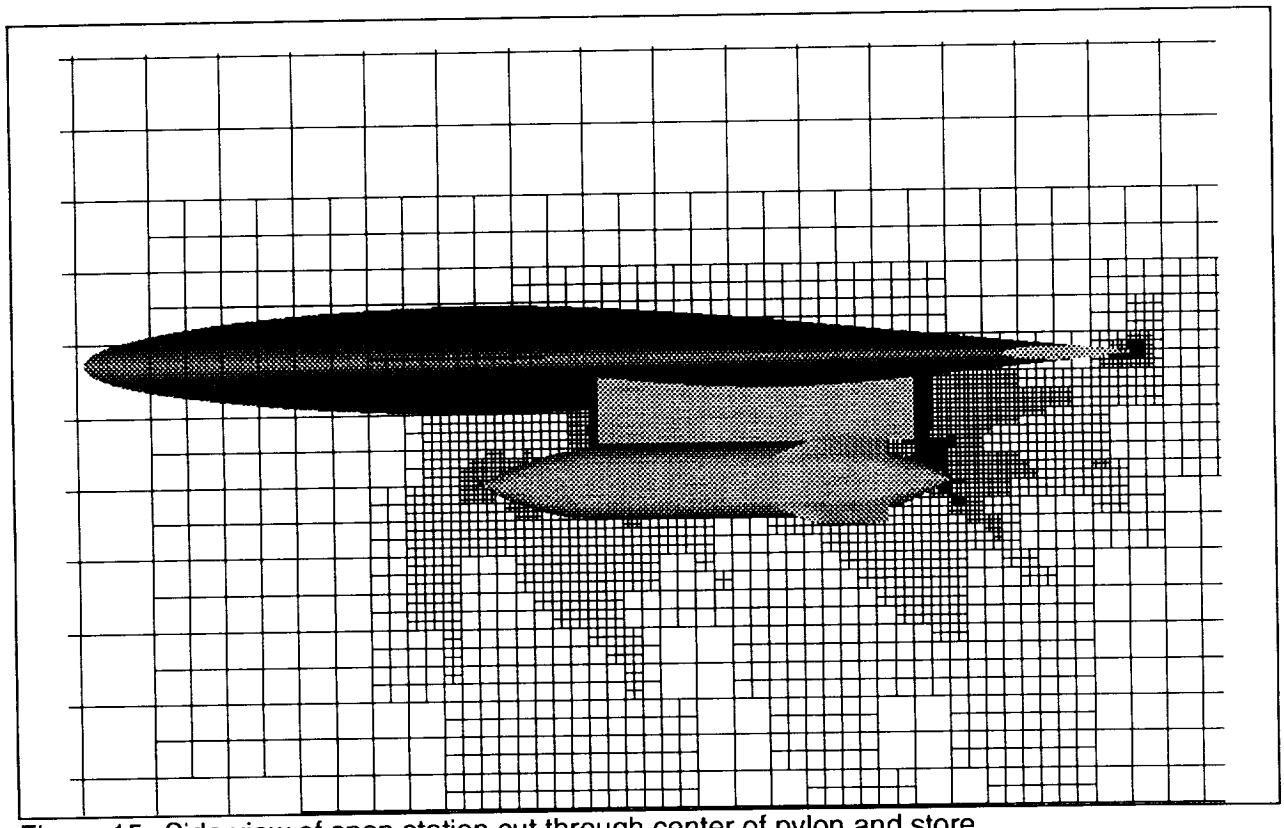


Figure 15. Side view of span station cut through center of pylon and store.

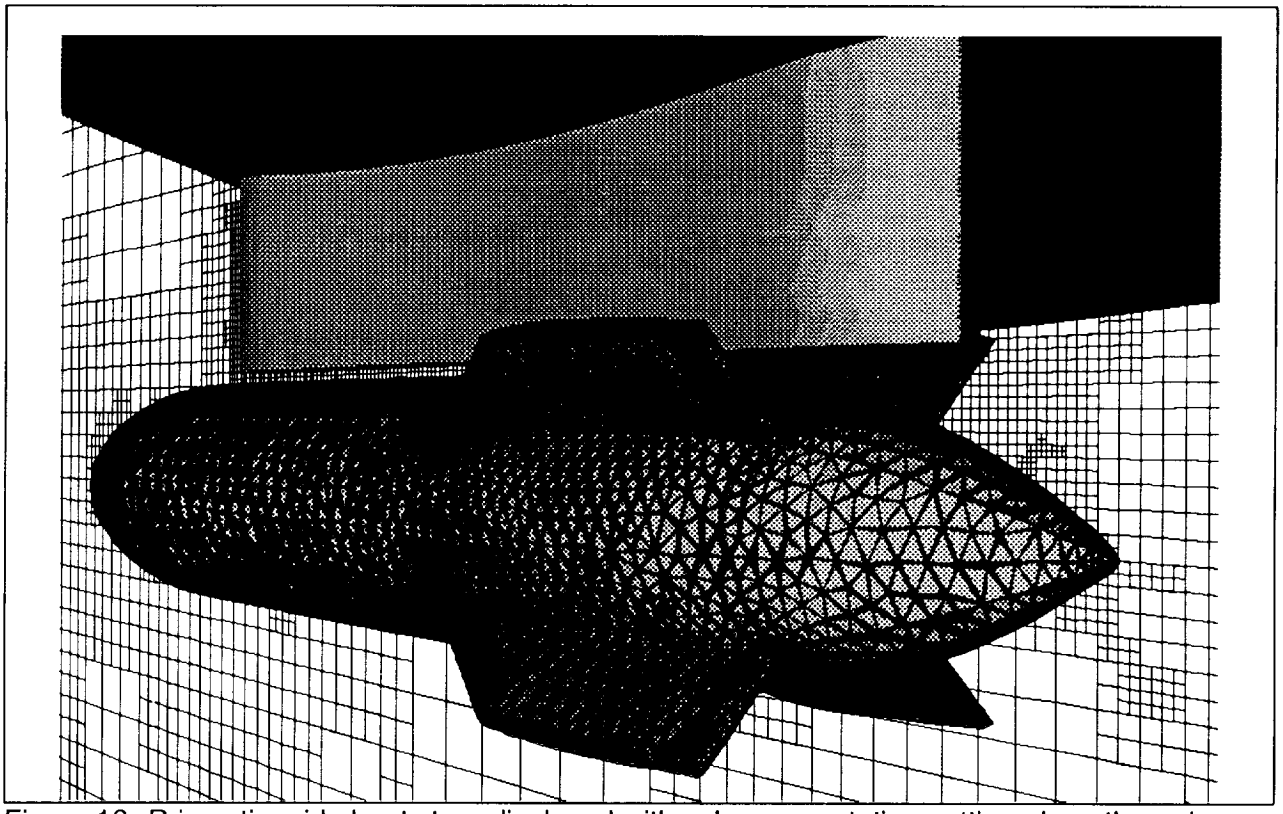


Figure 16. Prismatic grid about store displayed with pylon span station cutting plane through Cartesian grid.

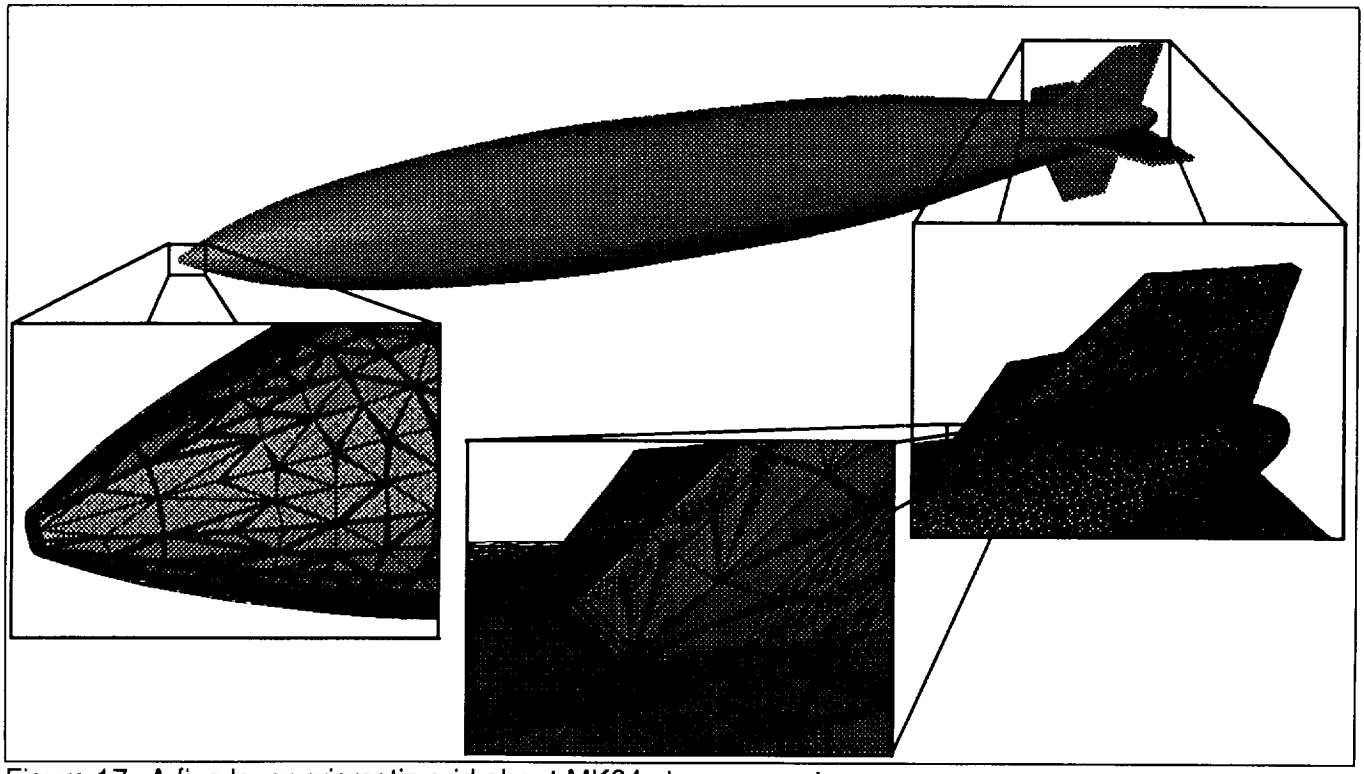


Figure 17. A five layer prismatic grid about MK84 store geometry.

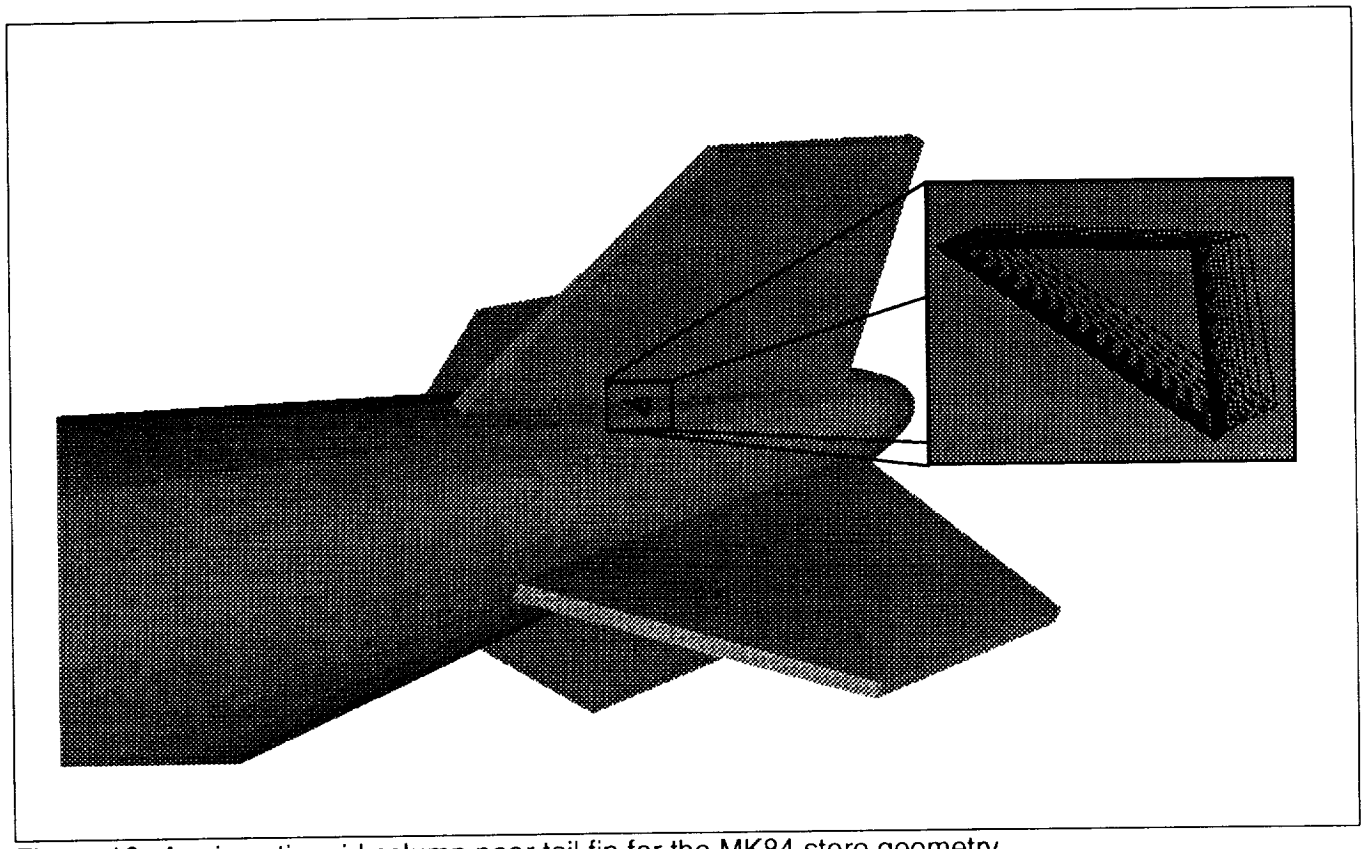


Figure 18. A prismatic grid column near tail fin for the MK84 store geometry.

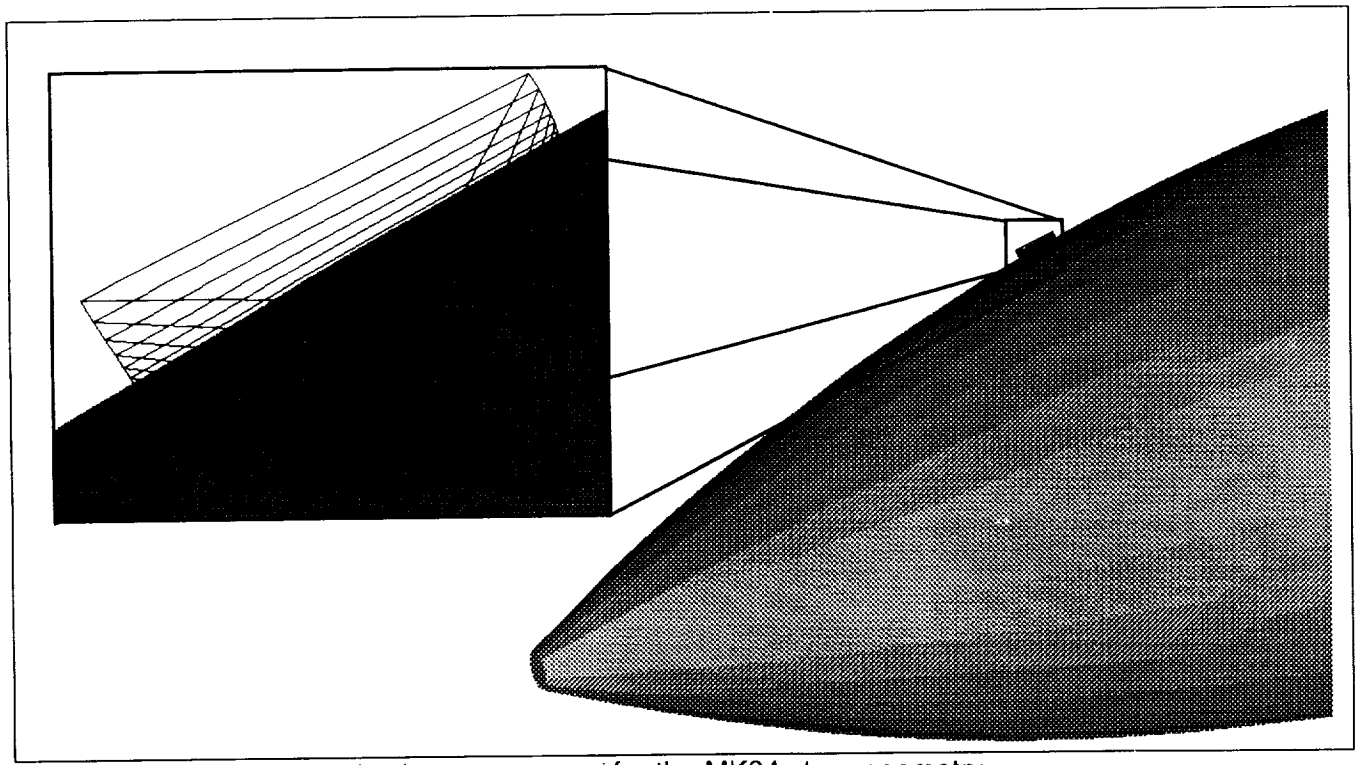


Figure 19. A prismatic grid column near nose for the MK84 store geometry.

# UNSTRUCTURED GRID TECHNOLOGY

19951223321

N95-28742

# PROSPECTS AND EXPECTATIONS FOR UNSTRUCTURED METHODS

Timothy J. Baker
Princeton University
Department of Mechanical and Aerospace Engineering
Princeton, New Jersey 08544 USA

## SUMMARY

The last decade has witnessed a vigorous and sustained research effort on unstructured methods for computational fluid dynamics. Unstructured mesh generators and flow solvers have evolved to the point where they are now in use for design purposes throughout the aerospace industry. In this paper we survey the various mesh types, structured as well as unstructured, and examine their relative strengths and weaknesses. We argue that unstructured methodology does offer the best prospect for the next generation of computational fluid dynamics algorithms.

## INTRODUCTION

Mesh generation has long been recognized as a major pacing item in computational fluid dynamics. But over the last decade, it has rapidly evolved into a broadly based area of research. Early meshes were almost exclusively quadrilateral or hexahedral, and were typically created by defining a suitable coordinate transformation from a unit square, or unit cube, into the physical domain. In this manner, one can create the familiar O, H and C-meshes around airfoils, wings and wing/fuselage combinations. These meshes inevitably possess a high degree of structure or regularity on account of their well defined set of coordinate directions. Regularity that is inherent in a structured mesh can, however, be a serious drawback when generating a mesh to conform with all boundaries of the domain. By the end of the seventies, it had become quite clear that this approach would not, by itself, suffice to handle complex configurations.

The formidable challenge presented by complex geometry has been met in various ways. The use of non-aligned meshes (ref. 31) circumvents the problem of conforming with boundaries at the cost of requiring interpolation formulae to impose the boundary conditions. A series of separate overlayed meshes, each of which conforms to the surface, represents an alternative approach that lifts the burden of interpolation from the boundaries to the flow domain (refs. 5 and 27).

To avoid interpolation issues, it is necessary to create a mesh that conforms with all boundaries and also maintains contiguity of meshlines. In the case of hexahedral meshes, this is best achieved by a multi-block technique which splits the flowfield into a number of blocks or subdomains (refs. 11, 17, 35 and 36). The splitting is chosen so that each subdomain can be covered by a structured hexahedral mesh with an associated set of coordinate directions. In general, the orientation of neighboring blocks will differ and the associated coordinate lines cannot be extended across block boundaries. A multi-block mesh can thus be thought of as structured at the level of an individual block, but unstructured when viewed globally as a collection of blocks. The introduction of paving techniques (refs. 6 and 39) takes this process one step further and generates a hexahedral mesh that can be thought of as entirely unstructured. In fact, one can interpret such a mesh as a multi-block mesh whose blocks are the individual hexahedra.

Triangular or tetrahedral meshes were for a long time the preserve of those in the finite element community. They were of little more than passing interest to developers of the finite difference and finite volume methods that were the main staple of research into computational fluid dynamics. The recent intense interest in tetrahedral meshes (refs. 1, 12, 18, 28, 37 and 38) and flow solvers is, however, an



PRECEDING PAGE BLANK NOT FILMED

acknowlegment of their utility and the relative ease with which tetrahedral meshes can be generated for complex shapes. The example shown in figure 1, of the flow solution over a supersonic transport, is typical of the type of problem that is now routinely handled by tetrahedral based methods (refs. 8 and 14).

In this paper we review the strengths and weaknesses of different mesh types in terms of ease mesh generation and the implications for flow algorithms. In particular, we consider the issues of flow adaption and viscous calculations. Both of these concerns must addressed to ensure accurate and reliable solutions for almost any fluid dynamics problem. On the basis of these considerations, we argue that unstructured meshes, using tetrahedra or a mixture of tetrahedra and prisms, do offer the best prospect for the next generation of computational fluid dynamics software.

## A TAXONOMY OF MESHES

The challenge of generating meshes for complex configurations has encouraged several innovative ideas for meshes based on hexahedra, tetrahedra as well as hybrid mesh types. Traditionally, the term structured has been associated with hexahedral meshes on account of the regularity that is inherent in any hexahedral mesh with a well defined set of coordinate directions. Since no global coordinates can be associated with a tetrahedral mesh, it was therefore natural to refer to these as unstructured meshes.

It is certainly apparent that a multi-block mesh is usually only structured at the block level. One can imagine increasing the number of blocks until each block degenerates into a single hexahedron and one has a fully unstructured hexahedral mesh. Examples of unstructured hexahedral meshes are those that are typically produced by paving techniques (refs. 6 and 39). Likewise, a hexahedral mesh that undergoes mesh enrichment (ref. 10), with the consequent addition of extra points on cell faces and edges, can no longer be thought of as structured. Indeed, flow solvers for such meshes rely on pointers and indirect addressing, and must thus maintain a data structure similiar to that of a tetrahedral flow solver. The more natural subdivision of mesh types is therefore into hexahedral, tetrahedral and hybrid (i.e. those using a mixture of element types). The adjectives "structured" and "unstructured" more properly refer to the degree of regularity whose presence can be discerned by a locally well defined set of coordinate directions.

Among hexahedral meshes there is a wide range from single block, structured to completely unstructured. In between these extremes are (i) the multi-block meshes which can be further subdivided into those with contiguous blocks and those whose blocks are overlayed (refs. 5 and 27), and (ii) non-aligned (i.e. non-boundary conforming) meshes (ref. 31). Contiguous multi-block meshes can be further subdivided into composite or patched, according to whether or not the mesh lines are continuous across block boundaries. Finally, the hybrid meshes include both mixed tetrahedral/hexahedral (refs. 26 and 33) and tetrahedral/prismatic (refs. 16 and 25), as well as the Cartesian meshes (refs. 9, 15, 23 and 32) whose cells are mostly hexahedra, but with some cells consisting of variously shaped polyhedra in regions adjacent to the domain boundaries. An attempt to catalog these mesh types is shown in figure 2.

## TRIANGULATION TECHNIQUES

There are three main approaches to tetrahedral mesh generation, (i) octree decomposition, (ii) Delaunay based methods, and (iii) moving front techniques. A brief outline of the differences between these approaches follows.

### Octree Decomposition

In two dimensions this procedure can be viewed as a division of the flowfield into a collection of rectangles followed by a division of rectangles into triangles. Rectangles can be further subdivided into

four new rectangles, corresponding to the addition of an extra vertex at the mid-point of each side of the original rectangle. For rectangles which intersect the boundary, this subdivision can be repeated until a sufficiently fine resolution has been achieved. The first division of the flowfield into rectangles may be regarded as level zero, the subsequent division as level 1 and so on. Thus the final mesh will contain rectangles at level zero in the flowfield and highly refined rectangles in the vicinity of solid boundaries. Some degree of graduation in the mesh refinement is obtained by requiring adjacent rectangles to differ by no more than one level of subdivision.

The next stage consists of examining those rectangles which intersect the boundaries and replacing each such rectangle by a polygon consisting of that part of the rectangle lying in the flowfield together with the part of the solid boundary that lies inside the rectangle. After this cutting procedure has been applied, the mesh consists of a combination of rectangles in the flowfield together with polygons adjacent to the boundaries. This mesh could serve as a Cartesian mesh. Alternatively, the rectangles and boundary polygons could be further subdivided into triangles to provide a triangulation of the flowfield.

The concept generalizes in an obvious way to three dimensions, although the cutting procedure at the boundaries becomes much more complicated (refs. 34 and 38). The main drawback of this approach, however, is the inability to match a prescribed surface triangulation. In addition, since each surface triangle arises from the intersection of a hexahedron with the boundary, it is not clear how one can control the variation in triangle size and shape.

## Delaunay Methods

The Delaunay triangulation provides a sound framework for tetrahedral mesh generation and several Delaunay based methods have been developed. Many of these methods (refs. 1, 12, 21, 29 and 37) exploit an incremental algorithm that starts with an initial triangulation of just a few points. The complete triangulation is generated by introducing a point and locally reconstructing the triangulation after each point insertion. Incremental algorithms can be used equally well for both the initial mesh algorithm and for any further enrichment that may be required by solution adaptive refinement. A particularly attractive feature of this approach is the opportunity to place new points at specified locations with the object of retaining, and in many cases improving, the quality of the mesh. Recent work in this area has produced encouraging results (refs. 3, 4, 29 and 37) showing that mesh quality can be controlled and meshes created to achieve a guaranteed level of quality according to a suitable set of criteria. Figure 3a shows an initial triangulation of a complex planar domain; figure 3b shows the result after selective refinement of this mesh by the so called Voronoi segment method (refs. 4 and 29).

The main difficulty, for any Delaunay method, is the need to ensure surface integrity. This usually requires the triangulation near the surface to be altered in some way by overriding the Delaunay algorithm. Most methods triangulate the entire domain, extracting the surface triangulation afterward. One possibility is to insert the surface points first, identify the tetrahedra which make up the object and then insert the flowfield points so that no flagged tetrahedra are removed (refs. 1 and 2). Other methods (refs. 12 and 37) allow the volume triangulation to proceed unchecked and then re-establish the surface edges and faces by a series of edge/face swaps and the occasional introduction of an extra point.

## Moving Front Methods

The main virtue of this approach (refs. 18 and 28) is that it starts from a prescribed boundary triangulation which remains intact throughout the mesh generation process. The boundary triangulation is regarded as a front on which a new layer of tetrahedra is built. As a result, the original front triangles

become interior faces of the mesh and a new set of front faces is created. The algorithm continues to build tetrahedra on the new front, growing more tetrahedra until the entire domain has been filled. A particular difficulty of this method occurs in the closing stages of the procedure when the front is folding in on itself and the final vestiges of empty space are replaced by tetrahedra. It is clearly necessary, in the final stages, to maintain good control over the size of the front faces as well as the shape of the unfilled domain that is left.

## MESH IMPLICATIONS FOR FLOW ALGORITHMS

Single block structured meshes, which dominated the early development of computational fluid dynamics, are well suited to exploitation by implicit algorithms (e.g. Alternating Direction Implicit methods) as well as mapping easily onto vector computer architectires. The later development of unstructured meshes forces a reappraisal of this situation and there is a consequent need for algorithms which work well on unstructured meshes. In fact, an explicit method appears to be the natural choice for solving the flow equations on an unstructured mesh and the availability of very efficient multigrid techniques removes the otherwise serious restriction of an explicit time step stability limit.

Even so, there is an overhead for the computational work associated with tetrahedral methods. A tetrahedral mesh of N points has roughly 6N cells, 12N faces and 7N edges. The fluxes can be accumulated either across faces, or alternatively, along edges. For tetrahedral meshes there is an obvious advantage in operation count to exploit edge based data structures. A mesh of hexahedra has roughly N cells, 3N faces and 3N edges, so that the operation count is comparable whether fluxes are accumulated across faces or along edges. Either way there is still a clear 2:1 advantage in computational efficiency for a hexahedral flow solver over a tetrahedral flow solver using an edge based data structure on meshes with the same number of points. This leaves open the question of whether comparable accuracy can be achieved on a tetrahedral mesh with fewer points. For a simple shape (e.g. a wing) with an optimal mesh distribution, one would expect comparable accuracy from both mesh types for a similar number of mesh points. For a more complex shape, the regularity inherent in a structured mesh can often lead to an unnecessary refinement in areas far removed from the boundary. In such cases, there is an inefficient distribution in the structured mesh with the possibility that comparable accuracy could be achieved on an unstructured mesh with fewer points. The relative advantage in computational efficiency that is enjoyed by structured methods is therefore not excessive.

In order to attain an optimal use of computer resources, some rearrangement and pre-sorting of a tetrahedral mesh is necessary. For any given tetrahedral cell, the addresses of its forming points may occupy widely separated positions in memory. This feature will cause a severe degradation on computers that depend on a small high speed memory cache. In addition, it is necessary to employ indirect addressing, with the result that each point will be referenced every time it appears as the vertex of a cell. Since many cells are incident at a given mesh point, each point will be referenced several times, and the possibility of vector dependency will inhibit vectorization by the compiler. It is possible to overcome this problem by first sorting the cells into groups so that no point is referenced more than once in each group. On can then override the compiler and force vectorization.

A recursive bi-section procedure can efficiently decompose a mesh into any number of sub-domains with almost identical numbers of cells and edges. This provides a particularly convenient decomposition of the mesh for parallel computer architectures (refs. 7 and 14). Further re-addressing and sorting within each subdomain leads to efficient vectorization and cache use on each of the individual processors.

For a structured multi-block mesh, it is usually necessary to assign whole blocks to an individual processors. Since individual blocks may vary greatly in size, this means that one processor may be assigned to one block while another processor is assigned to several small blocks. It is difficult in this situation to achieve very good load balancing and some compromise in parallel performance can be expected. In contrast, the subdivision of an unstructured mesh maps very readily onto a parallel architecture with nearly perfect load balancing.

An example of the parallel efficiency that can be achieved is shown in figure 4. The figure presents results of running an unstructured flow solver with a tetrahedral mesh on an IBM SP2 parallel system (ref. 14). There is almost perfect scalability for up to 64 processors.

## ADAPTIVE REFINEMENT

The use of p-refinement which increases the accuracy of the discrete approximation is particularly effective for elliptic problems where a high degree of smoothness is expected. For problems involving discontinuities and sharply defined features, the use of mesh movement (r-refinement) or mesh enrichment (h-refinement) appears to be more suitable. Mesh movement has the virtue of leaving the mesh size and connectivity intact. Thus, a partition of the mesh into subdomains remains well balanced and re-partitioning of the mesh is not required. However, the movement or repositioning of mesh points inevitably reduces the definition in some parts of the domain in order to increase the resolution elsewhere. Furthermore, moving mesh lines is an inherently perilous procedure and special care must be taken to avoid overlapping of mesh lines and the appearance of cells with negative volume.

The remaining alternative, mesh enrichment, is particularly attractive since the mesh is altered only in the region where greater resolution is needed. For a parallel implementation of the flow algorithm, this requires a new partitioning of the mesh to maintain good load balancing. The computational cost of re-partitioning and sorting will therefore influence how often the mesh should be refined over the course of a calculation.

For a hexahedral mesh, the application of mesh enrichment (refs. 9, 10 and 15) introduces new points on the edges and faces as well as the center of a hexahedron. At the interface between a refined cell and a non-refined cell, four refined faces will abut a non-refined cell face. This upsets the regularity of a structured mesh and imposes alterations (e.g. the use of pointers and indirect addressing) on the flow solver, causing the flow algorithm data structure to closely resemble that of a tetrahedral based flow solver.

For a triangular or tetrahedral mesh, the triangulation that results from mesh enrichment (refs. 19, 21 and 24) will consist of a new collection of triangles or tetrahedra. In this case, no modification of the flow solver is needed and a tetrahedral mesh thus provides a natural setting for mesh enrichment. This characteristic is one of the key advantages of an unstructured, and more particularly, a tetrahedral based method.

There are two main approaches to mesh enrichment. Edge bisection (ref. 19) (i.e. adding new mesh points at the mid-points of the edges of candidate tetrahedra) leads to refined cells of the same aspect ratio as the original cell. This method is simple to implement and can be applied to any triangular or tetrahedral mesh. However, although it preserves cell aspect ratio, addition of new points in this manner will not improve the mesh quality if the original mesh is extremely coarse with several bad aspect ratio cells. An alternative approach, as mentioned above, is available with a Delaunay based method (refs. 4, 21 and 24). The insertion of new points at specified locations and re-triangulation by a Delaunay algorithm provides a great deal of flexibility and control over mesh quality.

## NAVIER STOKES MESHES

There is ample empirical evidence to show that solutions of the Euler equations for inviscid flow can be carried out to a high degree of accuracy on a tetrahedral mesh. However, there is still considerable debate as to whether high aspect ratio triangular, or tetrahedral, cells can properly resolve a shear layer. In other words, can the Reynolds averaged Navier Stokes equations be solved with sufficient accuracy on a tetrahedral mesh?

It is known that the truncation error of a finite volume discretization depends on the shape of the control volume. In particular, a trapezoidal approximation for a cell vertex method though nominally second order, becomes only first order accurate unless the control volume possesses central symmetry (ref. 30). The effect is localized and it appears that the global solution error remains second order (ref. 13). For the Euler equations, the possible loss of second order accuracy in truncation error does not appear to be a serious problem. However, it is most likely that this will pose a serious problem when trying to resolve a boundary layer flow.

The control volume associated with a given point P corresponds to the boundary of the collection of cells incident at point P. In the planar case, the cells are triangles and the control volume is the polygon formed by the edges opposite P (figure 5). In three dimensions, the cells are tetrahedra and the control volume is the polyhedron formed by the faces opposite P. Although it is very difficult to ensure central symmetry, a good quality mesh should be close enough to this ideal for there to be little degradation in truncation error. In particular, this should be the case for an isotropic distribution of mesh points with low aspect ratio cells.

In the case of a highly stretched mesh, it is much more difficult to maintain a control volume that has nearly central symmetry. It seems plausible that one way to come close to this ideal, is to use a high degree of structure in the mesh. Thus, in a boundary layer, one could insert points normal to the surface, just as one does for a structured mesh. In two dimensions these points could be connected to form right triangles (see figures 5 and 6). The surface curvature would cause the "right angles" to be slightly greater or less than $90^o$, but for a continuously turning surface these deviations would be small. When the surface tangent is discontinuous, however, special treatment is needed. At a trailing edge, for example, a wake region of similarly constructed cells should be inserted. At a re-entrant corner, the thin layer Navier Stokes assumption breaks down and an isotropic mesh distribution is then appropriate.

The requirement of central symmetry imposes rigid constraints on the triangle connectivity. Figure 5 shows that central symmetry of the control volume occurs when the hypotenuse of each right triangle is oriented in the same direction. In figure 7, one of the diagonals has switched direction and the control volume is no longer centrally symmetric.

In three dimensions this issue becomes more problematic and it is unlikely that central symmetry of control volumes can be properly assured. However, it again seems plausible that a carefully constructed mesh of tetrahedra, each with three mutually orthogonal faces, is most likely to approach this ideal. Work along these lines has been presented by Pirzadeh (ref. 28) for the moving front method and by Marchant and Weatherill (ref. 20) and Marcum (ref. 22) for Delaunay based methods.

It remains to be seen whether accurate solutions of shear layer flows can be obtained for the Reynolds numbers of interest on meshes composed entirely of triangles or tetrahedra. An alternative approach is the use of a hybrid mesh. For example, one could use a quadrilateral, or hexahedral, mesh close to the surface and change to an unstructured mesh outside the viscous region. The disadvantage of this remedy

is that one is again faced with most of the restrictions that arise with structured meshes. Another version of the hybrid mesh that offers more of the flexibility one looks for in an unstructured method, is provided by using prisms in the boundary layer region. Work in this area has been pioneered by Nakahashi (ref. 25) and more recently by Kalinderis et al (ref. 16). There are still some disadvantages to this approach; the use of two different cell types complicates the flow solver and mesh adaption no longer has the simplicity available on purely tetrahedral meshes. However, this may be the price that one has to pay in order to obtain highly accurate Navier Stokes solutions.

## FUTURE DIRECTIONS

In assessing the current status of mesh generation methods and future developments, it seems likely that there will be an increasing emphasis on two key requirements. First, there is a strong drive towards methods that are easy to use and reliable (i.e. user friendly). Second, there is an increasing expectation that the computational results should be accurate, ideally to within some known tolerance. As improved computer hardware opens up the opportunity to attack increasingly complex problems, an effective response to both of these issues will become even more urgent.

Although often regarded as a separate entity, it is important to remember that the mesh generator is a central part of the computational environment, interfacing on one side (through the CAD system) with the surface definition of the geometry, and on the other side with the flow solver. The quality of the mesh directly influences the quality of the solution, unfortunately in ways that are usually difficult to quantify. Mesh quality in the volume mesh can be related to cell aspect ratio and size variation. But there is also the question of the degree to which the surface mesh matches the true surface. Surface definition is therefore an important concern. It is clear that the interface between a surface mesh generator and the CAD system should preserve the surface definition. However, it is also necessary to ensure that the CAD description is sufficiently well defined to meet the needs of aerodynamic flow calculations. For example, it is often the case that a CAD definition is provided by a series of patches which do not always abut correctly at their joins. Furthermore, for Navier Stokes computations, small perturbations in the surface definition could easily project a significant way into the boundary layer, wreaking havoc on a Navier Stokes mesh and the resulting flow prediction.

It is therefore necessary to (i) quantify the quality of the underlying surface, placing tolerances on what deviations are acceptable, (ii) obtain measures of how well a surface mesh matches the true surface definition, and (iii) generate a volume mesh that conforms to the surface mesh and meets a set of previously defined mesh quality measures. It is important that adaptive refinement should be carried out without the need for any user intervention. Moreover, the refined mesh should also meet the same mesh quality measures that are required for the original mesh.

In order to put together a seamless, user friendly software environment, a high degree of modularity with clean interfaces between each software module is desirable. First, one requires a link between the CAD system and the surface mesh generator. It is likely that the interface will need to carry out checks on the CAD surface definition, and possibly process the CAD data to obtain a surface description that meets the stringent requirements for computational aerodynamics. The surface mesh generator will require some input from the user to define the mesh density, perhaps to the extent of defining surface patches and indicating which parts of the surface require high resolution. However, the generation of the surface mesh to a specified quality, in terms of mesh aspect ratio and size variation, should be handled entirely automatically. It would appear that surface mesh generation can be carried out with comparable ease for either quadrilateral or triangular elements.

The volume mesh generator should create a mesh, entirely automatically, that matches the prescribed surface mesh. It is the opinion of the author that ease of volume mesh generation and adaptive refinement are key advantages that will continue to favor tetrahedral meshes over their hexahedral counterparts. For Navier Stokes computations, however, it is quite possible that some compromise must be made and tetrahedral elements will not be able to capture the features of shear layers with sufficient accuracy. The most promising combination in this case would be the use of a prismatic mesh in the shear layer with tetrahedra elsewhere. It seems likely, however, that purely tetrahedral meshes will be used in the near term for Navier Stokes calculations, since tetrahedral mesh generation has advanced to a high degree of automation. In the longer term, the demand for high accuracy will probably drive the development of methods based on a combination of prisms and tetrahedra for three dimensional Navier Stokes computations.

## REFERENCES


1. Baker, T.J.: Three Dimensional Mesh Generation by Triangulation of Arbitrary Point Sets, AIAA 8th Computational Fluid Dynamics Conference, Honolulu, Hawaii, June 1987, AIAA Paper 87-1124.

2. Baker, T.J.: Shape Reconstruction and Volume Meshing for Complex Solids, Int. J. Num. Meth. Eng., Vol. 32, No. 4, 1991, pp. 665-675.

3. Baker, T.J.: Point Placement and Control of Triangle Quality for Inviscid and Viscous Mesh Generation, Proc. 4th Int. Conference on Numerical Grid Generation, (edited by Weatherill), Swansea, UK, April 1994, pp. 137-149.

4. Baker, T.J.: Triangulations, Mesh Generation and Point Placement Strategies, in Computing the Future, (edited by D. Caughey), Cornell University, Ithaca, NY, November, 1994, pp. 101-115.

5. Benek, J.A.; Buning, P.G.; and Steger, J.L.: A 3-D Chimera Grid Embedding Technique, AIAA 7th Computational Fluid Dynamics Conference, Cincinnati, Ohio, June 1985, AIAA Paper 85-1523.

6. Blacker, T.D.; and Stephenson, M.B.: "Paving: A New Approach to Automated Quadrilateral Mesh Generation", Int. J. Num. Meth. Eng., Vol. 32, No. 4, 1991, pp. 811-847.

7. Cheng, W.; Jameson, A.; and Mitty, T.J.: AIRPLANE on the IBM Parallel System SP1, Parallel CFD '94 Conference, Kyoto, Japan, May 1994.

8. Cliff, S.; Hicks, R.; and Baker, T.J.: Design and Computational/Experimental Analysis of Low Sonic Boom Configurations, Proc. High Speed Research Sonic Boom Workshop, NASA Langley, June 1994.

9. Coirier, W.J.; and Powell, K.G.: A Cartesian, Cell-Based Approach for Adaptively-Refined Solutions of the Euler and Navier-Stokes Equations, AIAA Aerospace Sciences Meeting, Reno, NV, January, 1995, AIAA Paper 95-0566.

10. Davis, R.L.; and Dannenhoffer, J.F.: 3-D Adaptive Grid-Embedding Euler Technique, AIAA 31st Aerospace Sciences Meeting and Exhibit, Reno, NV, January 1993, AIAA Paper 93-0330.

11. Eiseman, P.R.; Cheng, Z.; and Häuser, J.: Applications of Multiblock Grid Generations with Automatic Zoning, Proc. 4th Int. Conference on Numerical Grid Generation, Swansea, (edited by Weatherill), UK, April 1994, pp. 123-134.

12. George, P.L.; Hecht, F.; and Saltel, E.: Constraint of the Boundary and Automatic Mesh Generation, Proc. 2nd Int. Conference on Numerical Grid Generation, Miami, FL, December 1988, pp. 589-597.

13. Giles, M.B.: Accuracy of Node-Based Solutions on Irregular Meshes, 11th Int. Conference on Numerical Methods in Fluid Dynamics, Williamsburg, VA, June 1988.

14. Jameson, A.; Cliff, S.; Thomas, S.; Baker, T.J.; and Cheng, W.: Supersonic Transport Design on the IBM Parallel System SP2, Proc. Computational Aerosciences Workshop, Santa Clara, CA, March 1995.

15. Karman, S.L.: SPLITFLOW: A 3-D Unstructured Cartesian/Prismatic Grid CFD Code for Complex Geometries, AIAA 33rd Aerospace Sciences Meeting, Reno, NV, January 1995, AIAA Paper 95-0343.

16. Kallinderis, Y.; and Ward, S.: Prismatic Grid Generation with an Efficient Algebraic Method for Aircraft Configurations, "AIAA 10th Applied Aerodynamics Conference, Palo Alto, CA, June 1992, AIAA Paper 92-2721.

17. Lee, K.D.; Huang, M.; Yu, N.J.; and Rubbert, P.E.: Grid Generation for General Three-Dimensional Configurations, Proc. NASA Langley Workshop on Numerical Grid Generation Techniques, (edited by Smith), October 1980.

18. Löhner: Generation of Three-Dimensional Unstructured Grids by the Advancing-Front Method, AIAA 26th Aerospace Sciences Meeting, Reno, NV, January 1988, AIAA Paper 88-0515.

19. Löhner: Adaptive H-Refinement on 3-D Unstructured Grids for Transient Problems, AIAA 27th Aerospace Sciences Meeting, Reno, NV, January 1989, AIAA Paper 89-0365.

20. Marchant, M.J.; and Weatherill, N.P.: Unstructured Grid Generation for Viscous Flow Simulations, Proc. 4th Int. Conference on Numerical Grid Generation,(edited by Weatherill), Swansea, U.K., April 1994, pp. 151-162.

21. Marcum, D.L.; and Weatherill, N.P.: A Procedure for Efficient Generation of Solution Adaptive Unstructured Grids, Proc. 4th Int. Conference on Numerical Grid Generation, (edited by Weatherill), Swansea, U.K., April 1994, pp. 639-650.

22. Marcum, D.L.: Generation of Unstructured Grids for Viscous Flow Applications, 33rd Aerospace Sciences Meeting, Reno, NV, January 1995, AIAA Paper 95-0212.

23. Melton, J.; Berger, M.; and Aftosmis, M.: 3-D Applications of a Cartesian Grid Euler Method, 33rd Aerospace Sciences Meeting, Reno, NV, January 1995, AIAA Paper 95-0853.

24. Mitty, T.J.; Jameson, A.; and Baker, T.J.: Solution of Three-Dimensional Supersonic Flowfields via Adapting Unstructured Meshes, Computers Fluids, Vol. 22, No. 2/3, 1993, pp. 271-283.

25. Nakahashi, K.: Computations of Three-Dimensional Navier-Stokes Equations by Using a Prismatic Mesh, Proc. 6th NAL Symposium on Aircraft Computational Aerodynamics, Tokyo, Japan, June 1988.

26. Nakahashi, K.; Obayashi, S.: Viscous Flow Computations Using a Composite Grid, AIAA 8th Computational Fluid Dynamics Conference, Honolulu, Hawaii, June 1987, AIAA Paper 87-1128.

27. Onslow, S.H.; Blaylock, T.A.; Albone, C.M.; and Hodges, J.: The FAME Mesh Generation System - its Synthesis, Flexibility and Capabilities, Proc. 4th Int. Conference on Numerical Grid Generation, (edited by Weatherill), Swansea, U.K., April 1994, pp. 259-269.

28. Pirzadeh, S.: Unstructured Viscous Mesh Generation by Advancing-layers Method, AIAA 11th Applied Aerodynamics Conf. AIAA Paper 93-3453, Monterey, August 1993.

29. Rebay, S.: Efficient Unstructured Mesh Generation by Means of Delaunay Triangulation and Bowyer-Watson Algorithm, J. Comp. Physics, Vol. 106, 1993, pp. 125-138.

30. Roe, P.L.: Error Estimates for Cell-Vertex Solutions of the Compressible Euler Equations, ICASE Report No. 87-6.

31. Samant, S.S.; Bussoletti, J.E.; Johnson, F.T.; Burkhart, R.H.; Everson, B.L.; Melvin, R.G.; Young, D.P.; Erickson, L.L.; Madson, M.D.; and Woo, A.C.: TRANAIR: A Computer Code for Transonic Analyses of Arbitrary Configurations, AIAA 25th Aerospace Sciences Meeting, Reno, NV, January 1987, AIAA paper 87-0034.

32. Schneiders, R.; Oberschelp, W.; Weiler, R.; Kopp, R.; Bünten, R.; and Franzke, M.: Automatic Generation of Hexahedral Element Meshes for the Simulation of Metal Forming Processes, Proc. 4th Int. Conference on Numerical Grid Generation, (edited by Weatherill), Swansea, U.K., April 1994, pp. 223-233.

33. Shaw, J.A.; Georgala, J.M.; Peace, A.J.; and Childs, P.N.: The Construction, Application and Interpretation of Three-Dimensional Hybrid Meshes, Proc. 3rd Int. Conference on Numerical Grid Generation, (edited by Arcilla), Barcelona, Spain, June 1991.

34. Shephard, M.S.; and Georges, M.K.: Automatic Three-Dimensional Mesh Generation by the Finite Octree Technique, Int. J. Num. Meth. Eng., Vol. 32, No. 4, 1991, pp. 709-747.

35. Steinbrenner, J.P.; Karman, S.L.; and J.B. Chawner: Generation of Multiple block Grids for Arbitrary 3-D Geometries, in Three Dimensional Grid Generation for Complex Configurations: Recent Progress, (edited by J.L. Steger and J.F. Thompson), 1988, AGARDograph No. 309.

36. Thompson, J.F.: A Composite Grid Generation Code for General 3-D Regions, AIAA 25th Aerospace Sciences Meeting, Reno, NV, January 1987, AIAA Paper 87-0275.

37. Weatherill, N.P.; and Hassan, O.: Efficient Three-Dimensional Grid Generation using the Delaunay Triangulation, Proc. 1st European Comp. Fluid Dynamics Conf., Brussels, September 1992.

38. Yerry, M.A.; and Shephard, M.S.: Automatic Three-Dimensional Mesh Generation by the Modified-Octree Technique, Int. J. Num. Meth. Eng., Vol. 20, 1984, pp. 1965-1990.

39. Zhu, J.Z.; Zienkiewicz, O.C.; Hinton, E.; and Wu, J.: A New Approach to the Development of Automatic Quadrilateral Mesh Generation, Int. J. Num. Meth. Eng., Vol. 32, No. 4, 1991, pp. 849-866.

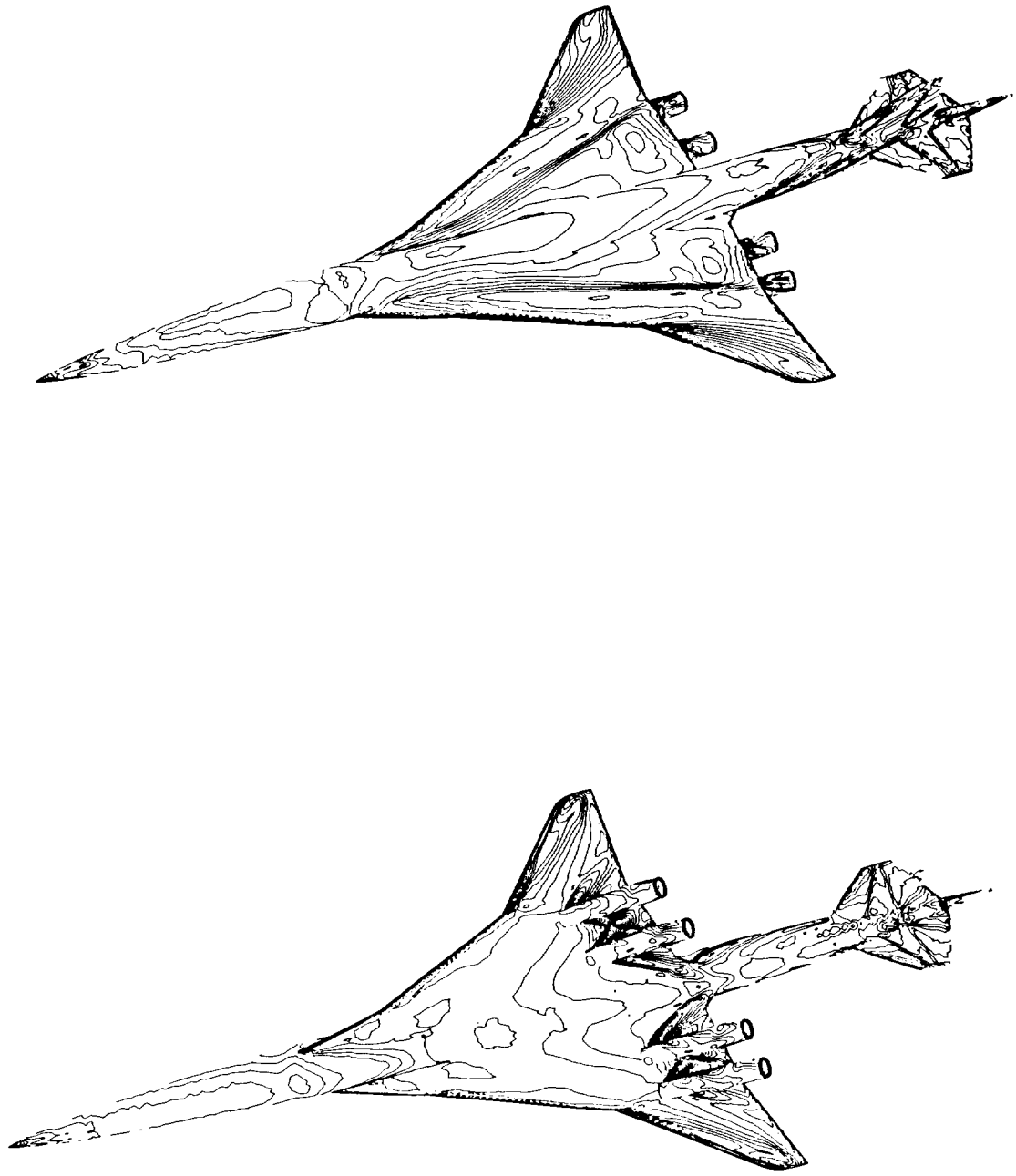


**Figure 1.-Surface Pressure Contours of the Flow Solution for a Supersonic Transport, $M_\infty = 2.4$, $\alpha = 3.6^o$**

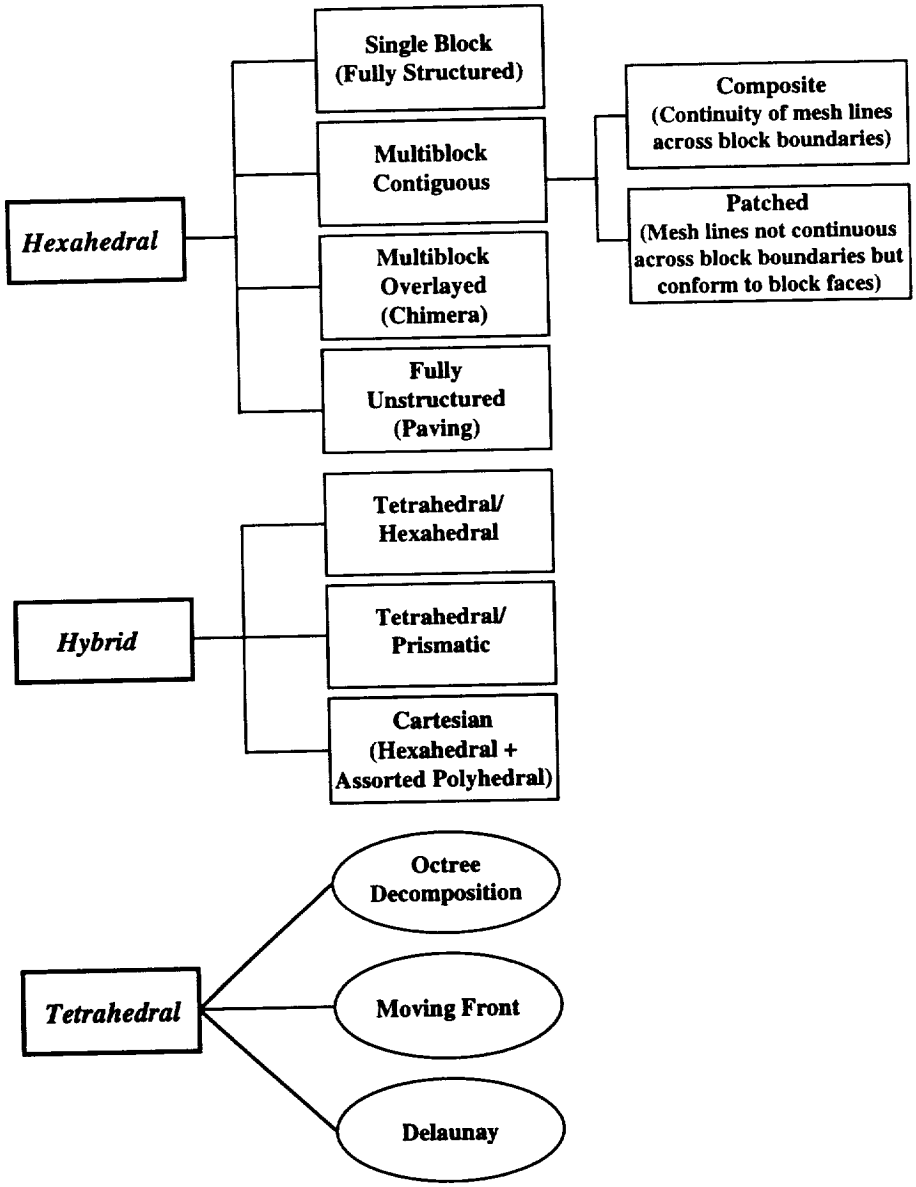



**Figure 2.- A Selection of Mesh Types**

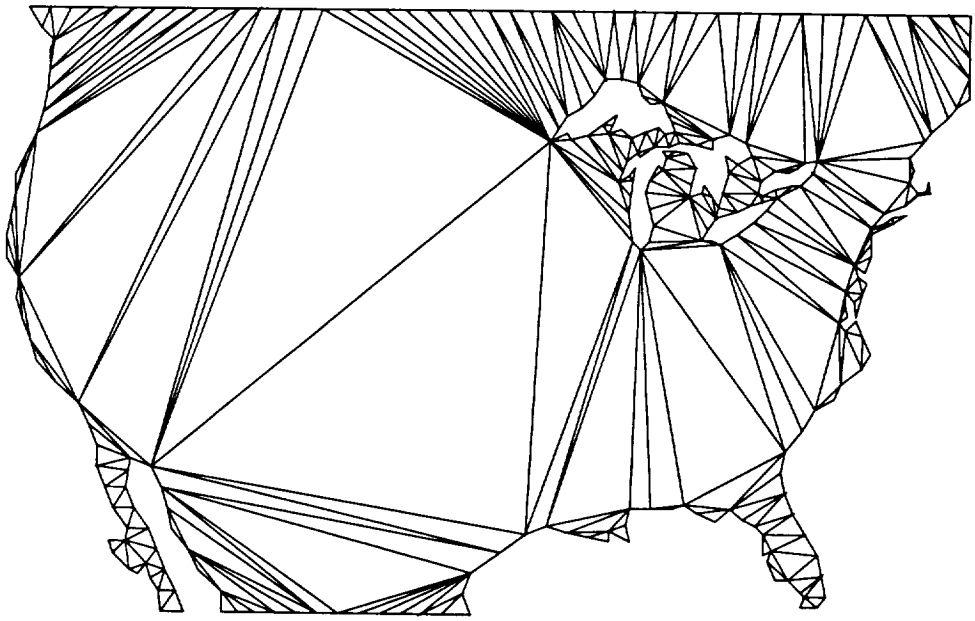


**Figure 3a.-Delaunay Triangulation of the Boundary Points**

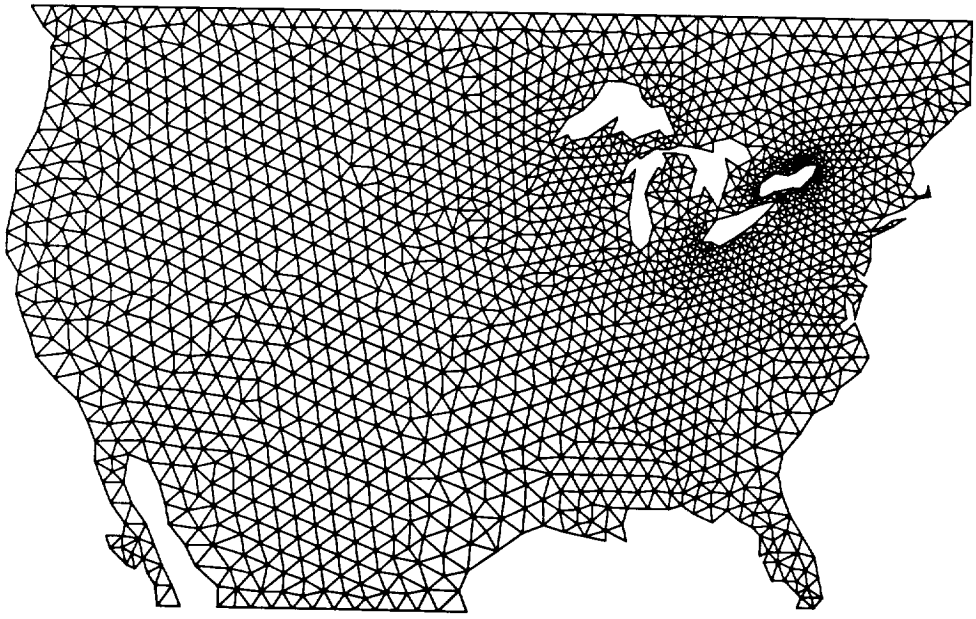


**Figure 3b.-Delaunay Triangulation of the Interior of the Domain Using Selective Refinement**

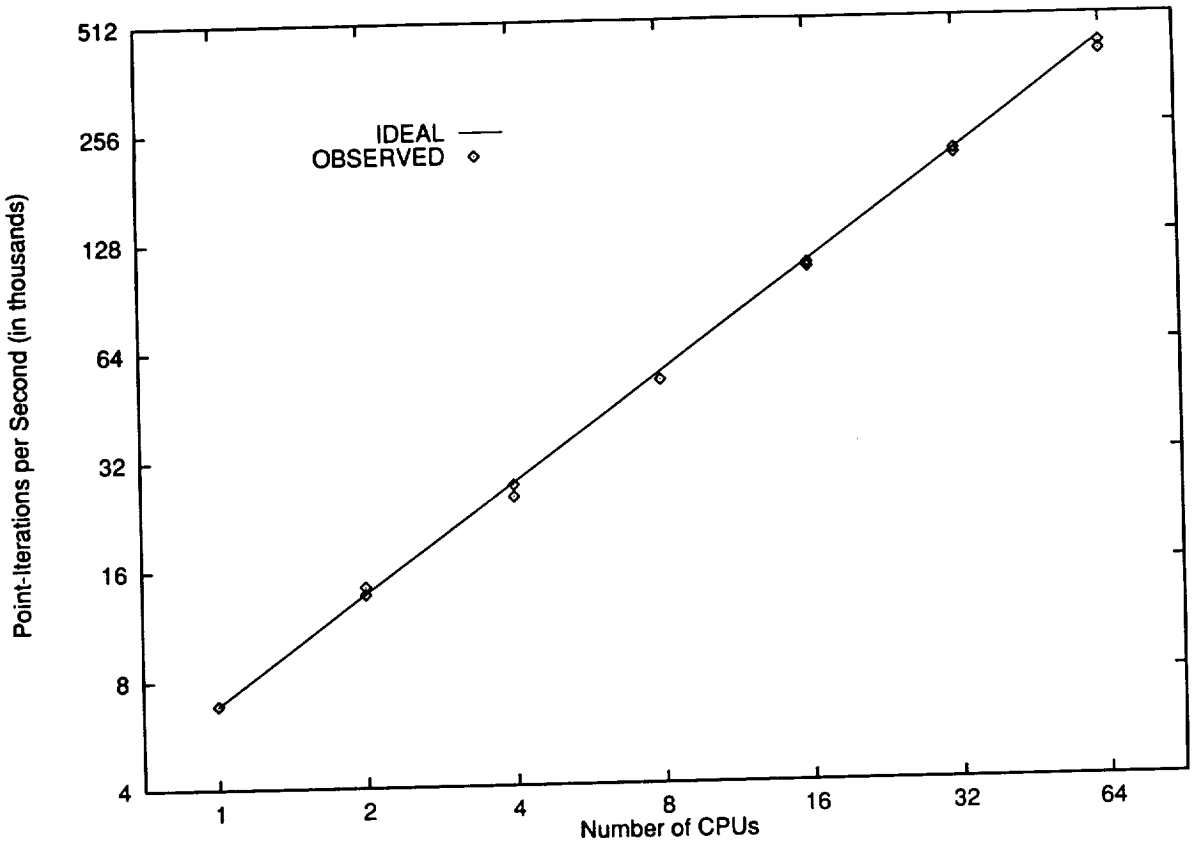



**Figure 4.-Parallel Efficiency of an Airplane Code on the IBM SP2 Parallel System**

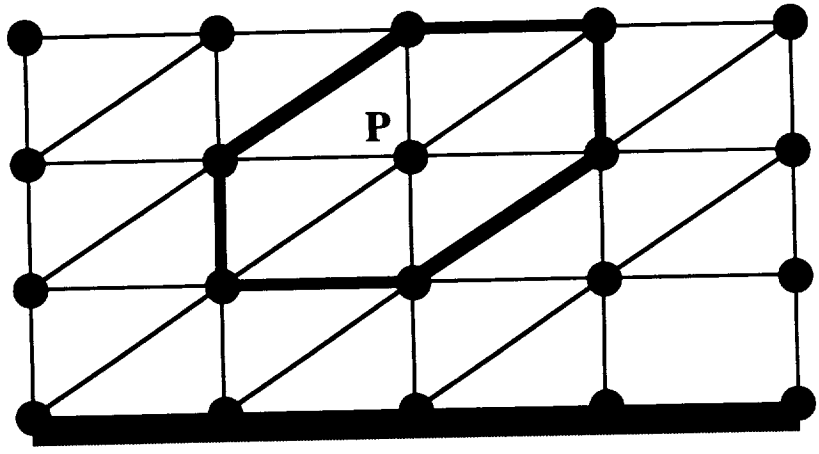



**Figure 5.-Control Volume for Point P has Central Symmetry**

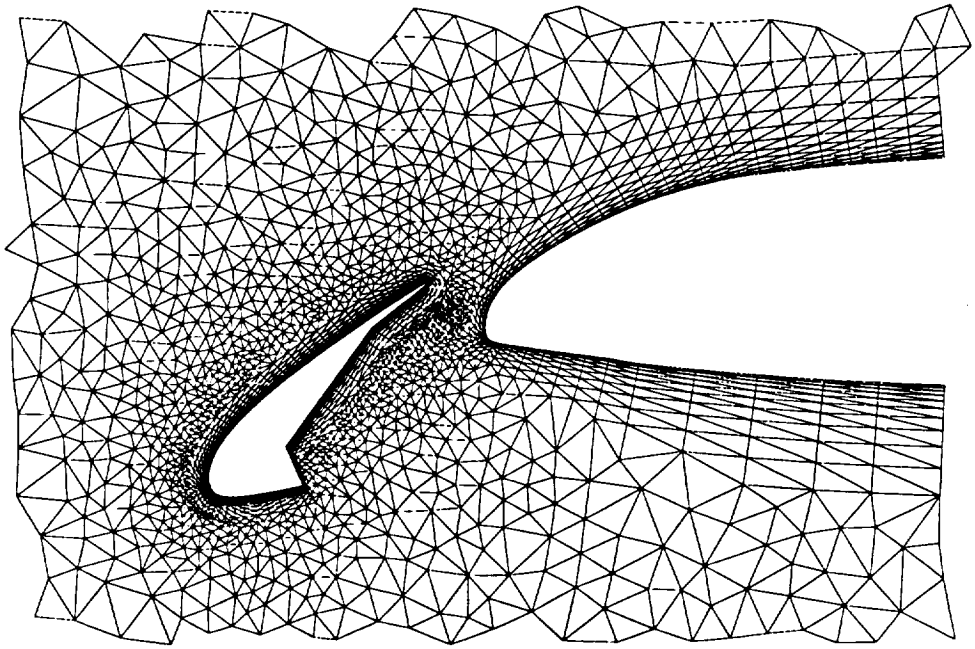


**Figure 6.-Triangulation of an Airfoil/Slat Region Suitable for Viscous Computation**

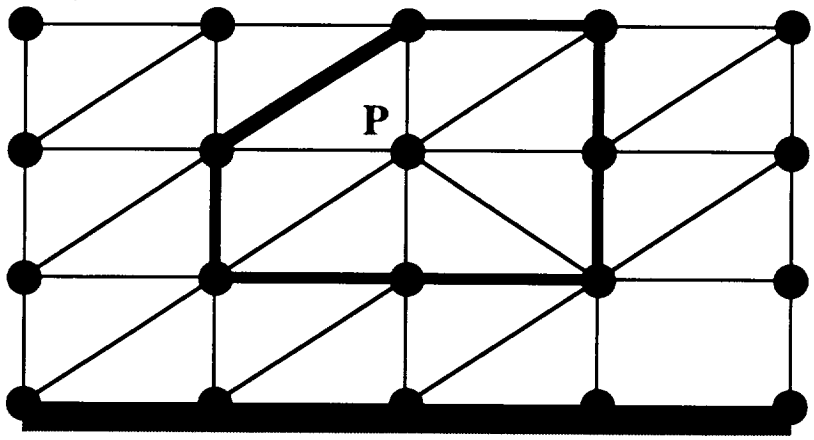



**Figure 7.-Control Volume for Point P does not have Central Symmetry**

N95- 28743

# AN UNSTRUCTURED-GRID SOFTWARE SYSTEM FOR SOLVING COMPLEX AERODYNAMIC PROBLEMS

Neal T. Frink
NASA Langley Research Center, Hampton, Virginia 23681

Shahyar Pirzadeh and Paresh Parikh[1]
ViGYAN, Inc., Hampton, Virginia 23666

## SUMMARY

A coordinated effort has been underway over the past four years to elevate unstructured-grid methodology to a mature level. The goal of this endeavor is to provide a validated capability to non-expert users for performing rapid aerodynamic analysis and design of complex configurations. The Euler component of the system is well developed, and is impacting a broad spectrum of engineering needs with capabilities such as rapid grid generation and inviscid flow analysis, inverse design, interactive boundary layers, and propulsion effects. Progress is also being made in the more tenuous Navier-Stokes component of the system. A robust grid generator is under development for constructing quality thin-layer tetrahedral grids, along with a companion Navier-Stokes flow solver. This paper presents an overview of this effort, along with a perspective on the present and future status of the methodology.

## INTRODUCTION

The role of advanced Computational Fluid Dynamics (CFD) is becoming increasingly important in the aerodynamic design process. The transition of CFD technology from the research lab to the project environment has progressed rapidly in recent years with encouraging results. However, the intense competitive pressures on U.S. industries to build more efficient or specialized products, e.g. aircraft, automobiles, buildings, pumps, heat exchangers, etc., in less time and at lower cost and risk continue to challenge the most ardent CFD method developer. The challenge now is to complete the transition and realize the full potential of CFD technology in the engineering environment. The driving issues affecting CFD utilization in such an environment are time, confidence, cost, and risk [1]. Thus, codes to be applied in the project environment are those which are validated, robust, readily usable by non-CFD expert engineers, and provide reliable, timely information at a reasonable cost.

The unstructured-grid (USG) approach has moved us much closer toward meeting these challenges. This methodology now enables modestly trained engineers to generate tetrahedral grids on complex configurations in a matter of days [2]. Inviscid flow solutions are readily obtainable from a number of good unstructured flow solvers [3]. Significant progress is also being made toward the grid generation and flow solution for viscous-flow problems [3], but additional work is needed before a reliable viscous USG capability emerges. This paper will include the status of our progress in this area.

The status of USG Euler methodology is fairly mature and can play a significant role in the preliminary design phase of aerodynamic products. This inviscid technology has proven to be quite valuable as an engineering 'tool' for providing critical insights which can lead to the timely solution of complex aerodynamic problems. For example, Ref. [4] describes a successful effort to redesign the wing/pylon intersection junction of a transport aircraft with under-wing nacelles by reducing the inviscid peak suction pressures in the region prone to viscous flow separation. The resulting drag reduction was verified by flight test. Work is also underway to assess the utility of this technology for modeling the underhood flow of automobiles.

1. Consultant



PRECEDING PAGE BLANK NOT FILMED

The utility of USG Euler technology has been further enhanced by coupling it with other established engineering tools. As will be described in the text, an inexpensive 'viscous' capability is provided by coupling the flow solver with an interactive boundary-layer method. A design capability is available with the addition of a powerful inverse design algorithm. Propulsion effects can be included through special boundary conditions. And while not discussed in this paper, work is underway to link a USG flow solver and grid generator with an aeroelastic module. When complete, an engineering capability should be available for the rapid design of aerodynamic surfaces for applicable complex configurations, which include the effects of propulsion flow, aeroelastic deformation, and a boundary layer.

There has been an ongoing coordinated effort at NASA LaRC over the past four years to mature the USG methodology. The goal is to provide a validated capability to non-expert users for performing rapid aerodynamic analysis and design of complex configurations. This has been accomplished by advancing the methodology in a team environment using an application-oriented approach where the method development/calibration was guided by parallel focused application assessments from selected users within both NASA and industry. The remainder of this paper will present an overview of our system, and a perspective on the present and future status of the technology.

## OVERVIEW OF CURRENT CAPABILITY

A diagram of the subject flow analysis system is presented in Figure 1. The primary components consist of a graphic user interface (GRIDTOOL), an advancing-front grid generator (VGRID), a cell-centered, upwind flow solver (USM3D), and a graphic post-processing code (VPLOT3D).

### Grid Generation

GRIDTOOL.—GRIDTOOL [5] is a graphic user interface (GUI) utility developed to serve as a "window" into the unstructured-grid generator, VGRID. The user must provide a surface definition in one of several formats, i.e. IGES, GRIDGEN, PLOT3D, LAWGS, or simple networks, which are converted within GRIDTOOL into a NURBS surface for manipulation. This GUI provides easy access to many functions, to be described below, which are useful in setting-up an input file for VGRID.

VGRID.—VGRID [2,6] is an interactive program for generating unstructured, tetrahedral/triangular grids by the advancing-front method [7]. A grid is generated by forming cells starting from the domain boundaries marching towards the interior of the computational domain. The advancing-front process is illustrated in Figure 2 for a simple 2D domain. The configuration of interest is first defined in terms of a number of surface patches (line segments in 2D). A transparent 'background grid' is used to control the distribution of points. The boundary segments are then triangulated (divided) individually to form the surface mesh or initial front. Next, tetrahedral (triangular) cells are generated over faces on the front by introducing new points in the field. During this process, old faces are replaced by new ones (reshaping the front), and the front is advanced in the field until the whole region is filled with grid cells.

For a 3D configuration, the geometry is divided into subdomains, using GRIDTOOL, on which 3- or 4-sided parametric surface patches are defined. The process of surface triangulation within VGRID involves transforming the 3D patch into a 2D parameter plane while preserving the overall projected shape of the 3D patch. After surface triangulation in 2D, the newly generated points are then transformed back onto the 3D patch by interpolating from its boundary curves. Depending on the patch topology, it is possible for points not to return exactly to the original surface after the transformation. As noted in Figure 1, the surface grid can optionally be passed through GRIDTOOL to project the points onto the surface.

The distribution of grid points is controlled by a 'structured' background grid [8]. This transparent grid consists of Cartesian mesh overlaying the entire domain upon which the user prescribes 'point' and 'line' sources to impose the desired spacing distribution. Parameters are available to control cell size, and the direction and intensity of spatial variation. A smooth variation of spacing is achieved throughout the computational domain by solv-

ing an elliptic partial differential equation on the Cartesian mesh. The approach is analogous to modeling heat diffusion from discrete heat sources in a conducting medium. A smooth variation of cell spacing is critical to the robustness of the grid generator and the quality of the grid.

The volume grid can be generated on a workstation through a series of restarts with VGRID. The code exploits a recurrent local/global renumbering system [2] which drastically reduces the computer memory requirement. In each run, a prescribed number of grid points and cells are generated. With the renumbering system, the point and cell numbers, in each restart run, start from one rather than the accumulated global numbers. This allows specification of relatively small array dimensions regardless of the final grid size.

After VGRID has completed the volume grid, there are usually some distorted tetrahedra, or some voids in the domain where the grid generator could not close the front. These problems occur when fronts approach from different directions in such a way that an ideal next point cannot be identified. As noted in Figure 1, the generated volume grid is postprocessed by the code POSTGRID [2] which removes distorted cells and/or a layer of cells around the problem areas, creates new fronts, and resumes the advancing-front process to close the grid. Since the modified front is now smoother after removing the anomalies, the problems are usually corrected, and a better quality tetrahedral mesh is produced.

The new versions of VGRID and POSTGRID incorporate recent improvements for better grid quality, robustness, and easier grid generation. The enhancements include improved graphics for better visualization of interactive surface and volume grid generation, graphic version of POSTGRID, interactive check and improvement of the surface and volume grid quality, dynamic memory allocation, better CPU efficiency, and many other improvements. VGRID requires about 80 bytes of memory per tetrahedron, and generates about 220 tetrahedra (including I/O) per second using a Silicon Graphics Indigo$^2$ workstation with R4000 processor.

## Flow Solver

USM3D.—USM3D [9-11] is a cell-centered, finite-volume upwind flow solver for solving the Euler equations on tetrahedral grids. Inviscid flux quantities are computed across each cell face using the Roe [12] flux-difference splitting approach (FDS), or the Van Leer [13] flux-vector splitting technique (FVS). Spatial discretization is accomplished by a novel cell reconstruction process, which is based on an analytical formulation for computing solution gradients within tetrahedral cells. Solutions are advanced in time by either a 3-stage Jameson-style Runge-Kutta explicit time-stepping scheme with convergence accelerated to steady state by local time stepping and implicit residual smoothing, or the linearized backward Euler implicit scheme of Ref. [14].

*Cell reconstruction scheme*: The higher-order reconstruction scheme, derived in Ref. [11,15] and illustrated in Figure 3, is based on a Taylor series expansion of the cell-averaged solution to the cell face. A key component of the scheme is the reconstruction of surrounding cell-averaged data to a common vertex or node by a weighted averaging procedure. Reference [9] proposed a scheme based on an inverse-distance weighted averaging of the primitive variables from the cell centroid to the cell vertices. While this approach has proven to be both accurate and robust through wide application, it is not fully second-order accurate in space. It has been shown in Ref. [16] to be approximately 1.85-order accurate.

As development efforts progressed toward solving the Navier-Stokes equations on highly stretched tetrahedral grids, it became evident that the accuracy of the inverse-distance averaging scheme was not adequate. Thus, a fully second-order accurate averaging procedure was implemented which is based on work by Holmes and Connell [17] and Rausch, et. al. [18]. The procedure is derived by solving a constrained minimization problem to determine weight factors which satisfy Laplacian relationships. The algorithm reconstructs to machine accuracy the exact values of a linear function at a node from surrounding cell-centered function values on an arbitrary tetrahedral grid. Furthermore, the simple universal formula shown in Figure 3 for expanding the cell-centered data to the cell faces also reconstructs the exact value of a linear function to the cell face. Thus, the entire spatial reconstruction scheme is termed second-order accurate, which has been verified by Mitchell [16].

*Limiter device*: Euler, finite-volume, upwind methods occasionally encounter circumstances where some local cell pressures, $p_{cell}$, become negative and preclude convergence of the global solution. This often occurs in regions where the inviscid assumptions are violated and viscosity would normally introduce enough diffusion to relieve the problem in real flow. Since the Euler equations are inviscid, the only source of diffusion in the solution comes from the discretization error inherent in the numerical approximation. A special empirical technique has been developed to resolve these local anomalies as they occur and permit the global solution to advance toward convergence. During each time iteration cycle, $p_{cell}$ is not permitted to become less than a user prescribed minimum value, $p_{tol}$, i.e. $p_{cell}=max(p_{cell},p_{tol})$. Furthermore, for higher-order differencing, cells which satisfy $p_{cell} < p_{tol}$ are set to and maintained with a more diffusive 1st-order stencil by imposing $\Phi_{cell}=0$, as shown in Figure 3. In many cases, this technique introduces enough local numerical diffusion to stabilize the global solution and permit convergence. It has greatly increased the robustness of USM3D for complex geometries.

*Data structure*: An efficient face-based data structure [19] is exploited in the flow code, which does not require interface fluxes to be stored during the solution cycle. The flux algorithm cycles through the triangular faces and accumulates the residuals by scattering each flux to the two adjacent tetrahedral cells. For proper vectorization, face-based data structure requires that the faces be ordered into groups or "colors" in which no two faces share the same cell or opposing node. The explicit time integration scheme requires 45 words/cell of memory and 18 μs/cell/cycle (higher order) on a Cray Y-MP supercomputer. Similarly, the implicit scheme requires 180 words/cell and 64 μs/cell/cycle (higher order) on the Cray Y-MP. While the implicit scheme requires more CPU time per iteration than the explicit scheme, it is demonstrated in Ref. [11] that the implict algorithm converges in 6-times less computer time than the explict one since higher Courant-Friedrichs-Lewy (CFL) numbers can be utilized. The following table shows a machine-to-machine comparison of Central Processing Units (CPU) requirements for USM3D on a test grid for the ONERA M6 wing. The computations involved a combination of 1st- and higher-order differencing. No special effort was made to enhance vectorization or performance on the non-Cray machines. The larger relative CPU time for the Iris Indigo$^2$ between implicit (98.9) and explicit (27.6) time integration is related to excessive swapping of the buffer due to insufficient core memory. Implicit calculations were not made on the Sun 4 because of insufficient memory.

| Computer | Explicit | | Implicit | |
|---|---|---|---|---|
| | CPU time, μs/cell/cycle | CPU time relative to Cray Y-MP | CPU time, μs/cell/cycle | CPU time relative to Cray Y-MP |
| Cray C-90 | 7.0 | 0.44 | 29.0 | 0.45 |
| Cray Y-MP | 16.2 | 1.0 | 64.0 | 1.0 |
| Convex | 331.5 | 20.5 | 1234.0 | 19.3 |
| Iris Indigo$^2$ (R4000) | 448.0 | 27.6 | 6335.0 | 98.9 |
| Sun 4 | 695.0 | 42.9 | ——— | ——— |

Boundary Conditions.—The USM3D code is designed for the easy addition/modification of boundary conditions (B.C.). It contains standard B.C.'s for flow tangency or no-slip on solid surfaces, characteristic inflow/outflow for subsonic outer boundaries, and freestream inflow and extrapolation outflow for supersonic flow. Some additional special boundary conditions, which have been added to improve the robustness of the code or to expand its capability, are described in the following.

*Blunt base*: With the application of USM3D to a wide range of complex 3D configurations, numerical difficulties routinely occur in blunt-based regions where a wake-type flow would develop in real flow, e.g. a blunt wing trailing edge or a blunt termination of a fuselage. For such cases, the inviscid flow assumption yields a singularity where the flow attempts to negotiate the corner.

A special boundary condition has been developed to mimic the relieving effect of a blunt-base wake on the surrounding inviscid flow field, which is to provide a smooth departure of the flow past the corner. As illustrated in Figure 4, the reconstructed primitive variables, $(\rho,u,v,w,p)$, from the nodes of blunt-base boundary cells are averaged and assigned directly to the boundary face and ghost cell. The effect is to introduce a *solution defined* transpiration boundary condition to the subject faces which simulates the presence of a wake.

*Engine model*: USM3D contains a feature for modeling propulsion-induced effects of up to four jet-engine or jet-nozzle components. The model [20] computes inflow and/or outflow boundary conditions from user prescribed values for the total pressure and temperature change across or within the combustor, the exit pressure, the fuel fraction, and the direction cosines for the jet. When applicable, the inlet mass flow is automatically balanced with that of the nozzle.

Preprocessing.—A preprocessing step is involved before running USM3D on a given grid generated by VGRID. The VGRID code outputs a coordinate file of the vertex locations, a cell-to-node connectivity file, and a triangle-to-patch correlation file. The PREFLO code, depicted in Figure 1 generates an additional face-to-cell connectivity file which is required for the face-based data structure in USM3D. (There are plans to modify the VGRID output to generate the appropriate files and eliminate this preprocessing step.) Boundary conditions are applied at the time of USM3D execution through a B.C. map file output by GRIDTOOL.

## Graphic Postprocessing, VPLOT3D

VPLOT3D is an interactive, menu-driven post processing program for extraction and display of data on unstructured (tetrahedral) grids. The philosophy behind the development of VPLOT3D has been to keep the program as simple and straight-forward as possible. There is only one 'text' window and only one main 'graphics display' window. Most of the interaction and steering takes place through pop-up menus.

VPLOT3D has the ability to display grids and flow quantities on either boundary surfaces or user defined arbitrary planes in the field. Flow quantities can be displayed using line or filled contours, velocity vectors and particle traces. Some unique features of the program include dynamic memory allocation, a 'probe' option that lets a user query the local value of a certain displayed fluid dynamic quantity, ability to plot two different data sets side-by-side and a journaling (log file) capability. Grid post-processing options allow a user to isolate and display 'bad' grid cells and to visually confirm boundary conditions before proceeding for flow solution. Several hardcopy output options are available including Black&White and Grayscale Postscript and RGB formats.

## TECHNOLOGY ASSESSMENT

The present USG flow analysis system is being widely used throughout industry and government laboratories. Studies have been published related to subsonic transports [4,21], high-speed research [22,23], military aircraft technology [24], turbomachinery applications [25,26], and code calibration/assessments [27-29]. Results from a published code calibration study are included herein to illustrate the capability.

A study was performed to assess the accuracy and utility of VGRID/USM3D on a wing/pylon/finned store configuration [27] shown in Figure 5. Tetrahedral grids were generated for the configuration with baseline (or store-in-carriage) store position, two dropped positions for $M_\infty=0.95$, and two positions for $M_\infty=1.2$. The initial grid was generated prior to the development of GRIDTOOL and took 4 days to complete. (The same task would require 1 or 2 days with present codes.) The remaining four grids were generated by 'freezing' the baseline surface triangulation while translating/rotating the surface patches, background-grid sources, and surface grid for the store to prescribed locations, then generating a new volume grid. The latter four grids required approximately one-hour each to generate.

Solutions were computed with Runge-Kutta time stepping and convergence acceleration using a CFL number of 4 in about 2000 iterations, while the L2 norm (rms average of all residuals) decreased by about 4 orders

of magnitude. For the baseline position, the computation required about 5.7h of Cray Y-MP time, whereas for the other two positions, the required computational time was of the order of 4 CPU h.

Figure 5 compares computed and experimental $C_p$ data [30] on the store along two streamwise rays (along top, $\phi$=5°, and bottom, $\phi$=185°) from the nose to the tail. For each case, the computed pressure coefficient compares very well with experimental data. In particular, note the comparison along the 5-deg ray for the baseline configuration that passes through the gap between the store and the pylon and where a large interference is expected. While not shown, very good agreement was also obtained on the fins of the store.

## STATUS OF EMERGING CAPABILITY

### Thin-Layer 'Viscous' Grid Generator

Advancing-Layers Method.—A new method for generation of thin-layer 'viscous' grids has been recently introduced and successfully applied to 2D [31] and 3D [32] problems. The method, referred to as the advancing-layers method (ALM), is based entirely on a modified advancing-front method (AFM) and benefits from the generality and flexibility of the conventional advancing-front-based Euler grid generators. Being based on an unstructured grid strategy, the method alleviates the structural limitations of the structured and semi-unstructured techniques.

The generation of a grid with the present approach is divided into three separate stages: 1) surface grid generation, 2) construction of high-aspect-ratio cells in the viscous region, and 3) generation of regular (isotropic) cells in the inviscid-flow region. Steps 1 and 3 utilize established methodology encompassed in the advancing-front inviscid grid generation code VGRID. The second step is performed independently by the ALM to reduce the overall complexity of the process. Although the generation of grid is divided into separate stages with the present approach, the entire process is performed in a single run with automatic transition from one stage to another.

The main features of the advancing-layers technique are similar to those of the conventional AFM. A volume grid is generated through a marching process in which tetrahedral cells originate from a triangular surface mesh and proceed into the computational domain. In contrast to the conventional method which adds cells in the field in no systematic sequence, the ALM advances one layer of cells at a time to reduce the complexity of producing high aspect ratio cells. Grid cells are formed by connecting new points, inserted along predetermined surface vectors using a prescribed stretching function, to the corresponding faces on the front. During the process, the integrity of the generated grid is monitored at each step. The layers continue to advance in the field, while growing in thickness, until either 1) opposite fronts approach to within a local cell size or 2) certain grid quality criteria, dictated by a global background grid, are locally satisfied. When the conditions are met on all faces on the front, the process automatically switches from the advancing-layers to the advancing-front mode to generate equilateral cells in the remaining inviscid-flow region.

To demonstrate the capability of the method, sample 2D and 3D grids around complex multi-component configurations are presented. The 2D geometry consists of three airfoil sections in a high-lift arrangement. The generated grid, shown in Figure 6, contains 34,987 cells and 17,705 points with a first layer spacing of about $7X10^{-6}$ main airfoil chord length. The flow solution, shown in Figure 6 was obtained on the grid at $M_\infty$=0.2, $a$=16.2°, and $Re_c$= $9X10^6$ using an available node-based, upwind flow solver [33] with the Baldwin-Barth turbulence model.

A representative complex 3D configuration was constructed to show the ability of the method to negotiate difficult aspects of 3D viscous grid generation. Although the geometry resembles a quasi-two-dimensional shape, it contains 3D complexities such as multiple components, sharp convex and concave edges, small gaps, and thin surfaces at the wing tip. The generated grid (surface and volume) is shown in Figure 7. The 'viscous' portion of the

grid contains 490,266 tetrahedral cells and has been generated using an SGI Indigo$^2$ workstation (R4000 processor) in only 4.3 minutes. The final grid contains a total of 168,734 points and 958,716 cells.

Anisotropic stretching:—In order to generate viscous grids of reasonable size on complex configurations, it will be necessary to have anisotropic (stretched) surface grids so that fewer points can be distributed in directions of reduced flow field gradients. The development of such a capability is currently under way which has produced satisfactory preliminary results. Figure 8 compares two surface grids for a wing configuration: one isotropic and another stretched along the leading and trailing edges of the wing. Also the field grid distributions are represented by triangulating a plane passing through the middle of the wing. The isotropic surface mesh (wing only) contains 40,778 equilateral triangles and is shown in Figure 8(a). A corresponding stretched grid with a similar chordwise grid distribution is shown in Figure 8(b). As indicated, the grid is stretched along the edges of the wing in different directions, with the stretching propagated in the field close to the body and smoothly dissipated further away from the geometry. The maximum cell aspect ratio in this grid is about 20:1 at the leading edge of the wing. Although the stretched grid has a similar distribution of points in the chordwise direction, it contains only 6,841 triangles on the wing.

## Navier-Stokes Flow Solver

Considerable progress has been made toward solving the Navier-Stokes equations on thin-layer tetrahedral grids with USM3D. Ref. [11] provided an initial validation of the solution algorithm on a tetrehedral grid constructed by subdividing the hexahedral cells of a structured grid. The following presents a full demonstration of computing a viscous-flow solution on a fully unstructured thin-layer tetrahedral grid produced from VGRID.

Hummel delta wing.—An unstructured viscous grid of 730,454 tetrahedral cells was generated for a delta wing of aspect-ratio 1 by an ALM version of the VGRID code. Normal spacing was distributed with geometric stretching near the surface to yield approximately 10 nodes (30 tetrahedra) in the mid-chord boundary layer for laminar flow. Anisotropic stretching, evident in the surface grid shown in the Figure 9, was utilized to reduce the chordwise density of cells in the grid—this feature resulted in a 7-fold reduction in grid size compared to standard isotropic triangles.

A viscous flow solution was computed with USM3D. A companion structured-grid computation was also performed with the CFL3D code [34] on a 65X65X33 (span-radial-chordwise) H-O grid for comparison. Both the unstructured and structured computations were performed on the delta wing at $M_\infty=0.3$, $a=20.5^\circ$, and $\mathrm{Re}_L=0.9X10^6$ with laminar flow.

The figure portrays the surface "oil-flow" pattern and a comparison of the spanwise distribution of pressure coefficient, $C_p$, at four chord stations. The coalescing and diverging streamlines in the computed flow patterns show evidence of the primary, secondary, and tertiary vortices. Good agreement is noted between the unstructured and structured solutions, and with the experimental data of Hummel [35].

Turbulence modeling.—The one equation turbulence model of Spalart-Allmaras [36] has been installed in USM3D and is presently undergoing testing. The derivation of this model relies heavily on empirical reasoning and dimensional analysis, and is receiving a growing acceptance within the CFD community. The turbulence equation solves for turbulent viscosity separately from the flow equations during each time step, resulting in a loosely coupled solution process. The governing equation is solved using the same backward-Euler time-stepping scheme [14] as used for the flow equations.

A preliminary assessment of the turbulence model has been made for flat-plate boundary layer flow. A two-dimensional flow was simulated by computing on a three-dimensional 'channel' grid which was constructed by subdividing the hexahedral cells of a structured grid into tetrahedra. The grid was sized for a $y^+\sim1$ and 20-25 nodes across the boundary layer with $M_\infty=0.5$ and $\mathrm{Re}_L=2X10^6$. Figure 10 depicts a very good agreement between the computed behavior of the law-of-the-wall and the model of Spalding [37]. Additional assessments are under-

way for a NACA 0012 airfoil and the ONERA M6 wing. The use of wall functions to eliminate the need to resolve the laminar sublayer is also under investigation.

## Interactive Boundary Layer

An interactive boundary layer (IBL) capability has been installed into USM3D [38] to provide for an inexpensive accounting of viscous effects in an unstructured-grid flow solver for predominantly attached flow cases, such as the transonic cruise condition of a subsonic transport. The two-dimensional strip boundary-layer method of Cebeci et al. [39] for turbulent, compressible flow was integrated with USM3D in a coupled fashion where the displacement thickness was simulated by transpiration velocities. The automated process begins with a partially converged inviscid solution and continues with the boundary layer method being invoked at specified intervals throughout the remainder of the solution iterations. The inviscid pressures along chordwise strips on the wing surface are passed to the boundary layer routine which calculates the estimated boundary layer displacement thickness. The transpiration velocities are computed along each strip from the boundary layer displacement thickness using Lighthill's equation [40]. A linear interpolation is performed between the strips to establish transpiration velocities at the remaining surface nodes. The face-based displacement boundary conditions are determined by averaging the node-based transpiration velocities to the centroid of the boundary triangle, then modifying the original tangent-flow condition to include the non-zero normal velocity. Face-center density and energy are set equal to their corresponding cell-center value. A similar capability for a node-centered unstructured flow solver is described in Ref. [41].

An application of the unstructured IBL capability, taken from Ref. [38], is shown in Figure 11 for a low-wing transport configuration with a supercritical airfoil at $M_\infty=0.77$, $a=1.12^\circ$, and $Re_{\bar{c}}=2.6X10^6$. The computational grid contains 286,000 tetrahedral cells and has ten chordwise strips along the wing span for applying the boundary layer method. A comparison of pressure coefficient at the 34-percent semispan station is shown in Figure 11. The inclusion of the boundary layer model results in substantial improvement in predicting both shock location and strength, as well as in the suction plateau. The poor agreement between the Euler result and the experimental data is typical of inviscid solutions on supercritical airfoils, where the shock location can be overpredicted by as much as 10- to 20-percent aft-chord.

## Inverse Iterative Design

The Constrained Direct Iterative Surface Curvature (CDISC) design method [42] has been coupled with USM3D to provide a capability for aerodynamic design of complex configurations. The CDISC method automatically generates and modifies the target pressures that are used by the basic DISC design module so that flow and geometry constraints are met. The CDISC module is coupled with the flow solver in an iterative manner and is called after a user-specified number of flow iterations (typically 25). After the target pressures are altered to meet the design constraints, the basic DISC module is used to modify the affected surface points. Geometry constraints are directly enforced on this new surface, then the volume grid is perturbed, using the spring analogy of Ref. [43], to reflect the geometry changes. Since the target pressure modification, surface design, and flow solution are converged in parallel, and the CDISC design system components themselves are very efficient, a converged design is usually obtained in about 25-50% more time beyond the initial baseline analysis.

To illustrate the USM3D/CDISC design capability, the wing of an executive transport configuration (Figure 12) has been redesigned for reduced drag at a primary design point without increasing the drag at a secondary design point. The baseline configuration was first analyzed using the USM3D code to establish reference values for performance comparisons at the primary (Mach = 0.8) and secondary (Mach = 0.9) design points. These conditions correspond to the long-range and high-speed cruise points, respectively. Two approaches are available in CDISC for reducing the wave drag of a configuration at transonic speeds: 1) a simple incremental method that limits the Mach number ahead of a shock to a specified value while matching lift and pitching-moment constraints; and 2) the Modified Uniform Distribution (MUD) method, which redistributes the chordwise loading over an airfoil section. The MUD option was selected for this case and was used to redesign the wing to reduce the drag at the pri-

mary design point. The wing planform, nacelle location, root airfoil section, airfoil maximum thickness, and spanwise load distribution were all held fixed. Initial results showed a 15 count drag improvement at Mach = 0.8, with only a 3 count penalty at Mach = 0.9. This design however had an unacceptable twist distribution just outboard of the nacelle. An input parameter to the MUD method was adjusted to shift the chordwise loading forward at several sections, which brought the twist distribution much closer to that of the original wing. The baseline and modified pressure distributions at Mach = 0.8, along with the corresponding airfoil sections, are shown in Figure 12 for two wing stations. At the inboard station, the strong shock near the quarter-chord of the baseline wing has been significantly weakened, resulting in an overall drag decrease of 6 counts. Analysis at the high-speed cruise point indicated that the new design had a 1 count drag improvement over the baseline at that condition. The design took an additional 4 hours of Cray C-90 time beyond the 16 hours required to obtain the initial baseline analysis. The grid had 700,000 tetrahedral cells, producing a central memory requirement of 148 megawords.

## Solution Adaptive Gridding

One of the strong advantages of unstructured-grid technology is the potential to exploit its flexibility in adapting or clustering the grid in the vicinity of dominant flow features, such as shocks and vortices, to enhance solution accuracy with fewer cells. This capability should facilitate a more rapid transfer of the technology to the workstation environment. Attention has been focused on providing an unstructured "remeshing" capability which distributes grid points efficiently throughout the domain and adequately resolves important flow features.

Remeshing procedures have been developed and applied for shock [44] and vortex [45] dominated flows. The procedure integrates the grid generation and flow solver functions in such a way that the information produced by one is successively used by the other until a desired accuracy is obtained. Accordingly, the grid generator VGRID, is combined with the flow solver USM3D, to produce a unified procedure that produces input parameters for the mesh generator based on the solution on a previous grid. The concept of a background grid, which is used in the advancing-front technique to specify and control grid size distribution, is exploited in the remeshing procedure.

Shocked flows:—The procedure of Ref. [44], which addressed shock dominated flows, was based on the "tetrahedral" background grid[1] concept [7] where the computational grid serves as the background grid for generating a subsequent grid. Grid spacing parameters were calculated at the nodes of this mesh based on the previous flow solution, and a new mesh was generated. Figure 13 shows the result of applying this remeshing procedure on the ONERA M6 wing. The upper portion of the figure compares the upper surface triangulation between the adapted grid (at the end of three adaptation cycles) and an unadapted grid. On the lower portion, $C_p$ distribution at two spanwise stations are compared (for $M_\infty=0.84$ and $a=3.06^\circ$) between the adapted and unadapted grids, and the experimental data. While both sets of computed data compare well with the experimental values, the efficiency of the adaptive procedure is established by the fact that the adapted grid had about 2.5 times fewer cells than the unadapted case.

Vortex flows:—A solution adaptive remeshing procedure, based on the "structured" Cartesian background grid concept, has been developed for low-speed vortex dominated flows [45], such as those produced by sharp swept-back leading edges or shed from the free ends of lifting surfaces. As discussed earlier with the structured background grid, grid spacing is controlled by point and line 'sources' strategically placed throughout the domain by the user. The present approach provides a capability for automatically prescribing line sources for refining the grid in regions of vortex flow.

In the current procedure, the vortices are located by an entropy parameter. In addition, a vortex in close proximity to a surface induces a static pressure peak on the surface. To accurately resolve both the vortex core and the accompanying static pressure peak, line sources are placed in the field and on the surface of the configuration. The procedure is automated to the point that the user need only specify the approximate starting locations for the

1. The "tetrahedral" background grid is no longer used in VGRID, but has been replaced by the "structured" background grid approach.

vortices and threshold cut-off values for both the entropy and the static pressure. All other functions, such as identifying the location and strength of the background grid 'sources' and creation of a new input for the grid generator, are automated.

The adaptation procedure has been applied to the Modular Transonic Vortex Interaction (MTVI) model, tested at NASA Langley, which is characterized by the presence of multiple vortices (see Figure 14). Computations were initiated on a relatively coarse grid with 35,388 points and 188,304 tetrahedral cells (GRID 1). Two successively finer adapted grids were generated with the adaptive procedure. The final adapted grid (GRID 3) had 68,158 points and 371,360 cells.

The left side of Figure 14 shows a comparison of surface triangulation for the unadapted (GRID 1) and the adapted (GRID 3) cases. For the adapted case, surface grid clustering near leading edges as well as near the wing tip is evident representing the static pressure peak due to proximity of vortices to the configuration surface. The bottom right portion of the figure shows a comparison of the field grid projected on to a cross sectional plane at 93-percent of root chord. Grid clustering around the vortex cores between the fuselage and the vertical tail and near the wing tip can be seen clearly. On the top right of the figure, flow field results are compared between the unadapted (GRID 1), the adapted (GRID 3) and experimental data. The $C_p$ comparison is shown at a streamwise station located at 93-percent of the root chord for $M_\infty=0.4$ and $a=10.54^o$. Results are also compared from a 'fine' grid with 148,285 points and 825,469 tetrahedra . During generation of the 'fine' grid, no special effort was made to cluster grid points around vortex core locations. An expected higher suction peak resulting from grid adaptation is evident. The fine grid did not capture the suction peak at the station shown due to lack of grid resolution there, while the adapted grid automatically provided the grid resolution needed. This case clearly establishes the ability of the adaptation procedure to automatically provide adequate grid resolution where needed. The whole adaptation process required about 2.5 hours of CPU time on a Cray-YMP computer, compared to about 4.0 hours for the 'fine' grid. The adapted results show a better accuracy with about 2.2 times less number of cells and about 38-percent CPU saving compared to the 'fine' grid case.

## CONCLUDING REMARKS

An overview has been presented of a maturing capability for performing rapid aerodynamic analysis and design of complex configurations using unstructured-grid methodology. Development priorities have been dictated by program needs from within NASA as well as those of industry partners. This technology also offers strong potential for impacting other non-aerospace disciplines, such as automobiles, boat sail and hull design, wind loads on buildings, design of ventilation systems, analysis of pumps and pipe systems, design of heat exchangers, and electromagnetics. The present unstructured Euler technology can be readily applied by modestly trained engineers to solve complex aerodynamic problems. Significant progress has also been made towards developing an unstructured Navier-Stokes capability for complex geometries.

Considerable utility can be derived from the base Euler technology. Often, the inviscid flow equations can provide the critical aerodynamic insights needed to resolve more complex problems. The Euler methodology has also been enhanced with additional engineering features such as interacting boundary layers, constrained inverse design, propulsion flow boundary conditions, and solution-adaptive gridding, thereby providing a powerful capability for aerodynamic design and analysis to the project-level engineer.

The capability for solving the Navier-Stokes equations on complex configurations using unstructured-grid methodology is still evolving. Such a flow solver has been developed and is presently being tested. The primary obstacle to a practical unstructured Navier-Stokes capability has been the lack of a robust thin-layer 'viscous' tetrahedral grid generator. As demonstrated in this paper, the Advancing-Layers method shows great promise toward providing that capability.

## ACKNOWLEDGMENTS

Many of the new capabilities presented herein reflect the contributions of several talented people to whom the authors are profoundly grateful. **Dr. W. Kyle Anderson** of the Computational Aerodynamics Branch at Langley provided generous assistance with installing his backward Euler time-implicit subroutines into USM3D, as well as providing technical advise on installing the 1-equation turbulence model in the code. **Dr. Peter M. Hartwich,** now with McDonnell Douglas Aerospace - West, integrated the engine model capability into the flow solver. **Ms. Leigh A. Smith** and **Mr. Richard L. Campbell** of the Transonic/Supersonic Aerodynamics Branch (T/SAB) at Langley contributed the work and text on the CDISC inverse-design method. **Mr. William D. Smith** and **Mr. Jassim A. Al-Saadi,** also of T/SAB contributed the IBL material.

## REFERENCES

1. Cosner, R. R. : "Issues in Aerospace Application of CFD Analysis", AIAA Paper 94-0464, January 1994.

2. Pirzadeh, S. : "Recent Progress in Unstructured Grid Generation", AIAA Paper 92-0445, January 1992.

3. Venkatakrishnan, V. : "A Perspective on Unstructured Grid Flow Solvers", AIAA Paper 95-0667, January 1995.

4. Potsdam, M. A., Intemann, G. A., Frink, N. T., Campbell, R. L., Smith, L. A., and Pirzadeh, S.: "Wing/Pylon Fillet Design Using Unstructured Mesh Euler Solvers", AIAA 93-3500, August, 1993.

5. Samareh-Abolhassani, Jamshid, "GridTool: A Tool for Surface Modeling and Grid Generation", *Proceedings of Workshop on Surface Modeling, Grid Generation, and Related Issues in CFD Solutions,* NASA CP-3291, May 9-11, 1995.

6. Parikh, P., Pirzadeh, S., and Löhner, R. : "A Package for 3-D Unstructured Grid Generation, Finite-Element Flow Solutions, and Flow-Field Visualization", NASA CR-182090, September 1990.

7. Löhner, R., and Parikh, P. : "Three-Dimensional Grid Generation by the Advancing Front Method", *Int. J. Num. Meth. Fluids*, 8, 1988, pp. 1135-1149.

8. Pirzadeh, S. : "Structured Background Grids for Generation of Unstructured Grids by Advancing-Front Method", *AIAA Journal*, Vol. 31, No. 2, pp. 257-265, February 1993.

9. Frink, N. T. : "Upwind Scheme for Solving the Euler Equations on Unstructured Tetrahedral Meshes", AIAA Journal, Vol. 30, No. 1, January 1992, pp. 70-77.

10. Frink, N. T., Parikh, P., and Pirzadeh, S. : "A Fast Upwind Solver for the Euler Equations on Three-Dimensional Unstructured Meshes", AIAA 91-0102, January 1991.

11. Frink, N. T. : "Recent Progress Toward a Three-Dimensional Unstructured Navier-Stokes Solver", AIAA Paper 94-0061, January 1994.

12. Roe, P. L. : "Characteristic Based Schemes for the Euler Equations", *Annual Review of Fluid Mechanics,* Vol. 18, 1986, pp. 337-365.

13. Van Leer, B. : "Flux-Vector Splitting for the Euler Equations", *Eighth International Conference on Numerical Methods in Fluid Dynamics*, E. Krause, ed., *Volume 170 of lecture Notes in Physics,* Springer-Verlag, 1982, pp. 507-512.

14. Anderson, W. K. : "Grid Generation and Flow Solution Method for Euler Equations on Unstructured Grids", NASA TM-4295, April 1992.

15. Frink, N. T. : "Three-Dimensional Upwind Scheme for Solving the Euler Equations on Unstructured Tetrahedral Grids", Ph. D. Dissertation, Virginia Polytechnic Institute and State University September 1991.

16. Mitchell, C. R. : "Improved Reconstruction Schemes for the Navier-Stokes Equations on Unstructured Meshes", AIAA 94-0642, January 1994.

17. Holmes, D. G. and Connell, S. D. : "Solution of the 2D Navier-Stokes Equations on Unstructured Adaptive Grids", presented at the *AIAA 9th Computational Fluid Dynamics Conference,* June 1989.

18. Rausch, R. D., Batina, J. T., and Yang, H. T. Y. : "Spatial Adaption Procedures on Unstructured Meshes For Accurate Unsteady Aerodynamic Flow Computation", AIAA Paper 91-1106, April 1991.

19. Jameson, A. and Baker, T. J. : "Improvements to the Aircraft Euler Method", AIAA Paper 87-0452, January 1987.

20. Hartwich, P. M. and Frink, N. T. : "Estimation of Propulsion-Induced Effects on Transonic Flows Over a Hypersonic Configuration", AIAA Paper 92-0523, January 1992.

21. Dodbele, S. S.: "Three Dimensional Aerodynamic Analysis of a High-Lift Transport Configuration", AIAA 93-3536, August, 1993.

22. Fouladi, K.: "Unstructured Grids for Sonic-Boom Analysis", AIAA 93-2929, July, 1993.

23. Carlsen, W. D.: "Development of Transonic Area-Rule Methodology", AIAA 94-1841, June 1994.

24. Wai, J., Herling, W. W., and Muilenburg, D. A., "Analysis of a Joined-Wing Configuration", AIAA 94-0657, January 1994.

25. Kwon, O. J., and Hah, C.: "Three-Dimensional Unstructured Grid Euler Method Applied to Turbine Blades", AIAA 93-0196, January, 1993.

26.Kwon, O. J. : "Solution of the 3-D Navier-Stokes Equations with a Two-Equation Turbulence Model on Unstructured Meshes Applied to Turbomachinery", AIAA Paper 94-1833, June 1994.

27. Parikh, P., Pirzadeh, S., and Frink, N. T. : "Unstructured Grid Solutions to a Wing/Pylon/Store Configuration", *Journal of Aircraft*, Vol. 31, No. 6, pp. 1291-1296, November-December 1994.

28. Ghaffari, F.: "On The Vortical-Flow Prediction Capability Of An Unstructured Grid Euler Solver", AIAA 94-0163, January, 1994.

29. Kinard, T. A., and Harris, B. W.: "Evaluation of Two Unstructured CFD Methods", AIAA 94-1877, June 1994.

30. Heim, E. R. : "CFD Wing/Pylon/Finned Store Mutual Interference Wind Tunnel Experiment", Arnold Engineering Development Center, AEDC-TSR-91-P4, January 1991

31. Pirzadeh, S. : "Unstructured Viscous Grid Generation by Advancing-Layers Method", AIAA Paper 93-3453, August, 1993.

32. Pirzadeh, S. : "Viscous Unstructured Three-Dimensional Grids by the Advancing-Layers Method", AIAA Paper No. 94-0417, January 1994.

33. Anderson, W. K. and Bonhaus, D. L. : "An Implicit Upwind Algorithm For Computing Turbulent Flows On Unstructured Grids", *Computers Fluids*, Vol. 23, No. 1, pp. 1-21, 1994.

34. Thomas, J. L., Taylor, S. L., and Anderson, W. K. : "Navier-Stokes Computations of Vortical Flows Over Low Aspect Ratio Wings", AIAA Paper 87-0207, January 1987.

35. Hummel, D. : "On the Vortex Formation Over a Slender Wing at Large Angles of Incidence", AGARD CP-247, Paper No. 15, February 1983.

36. Spalart, P. R. and Allmaras, S. R. : "A One-Equation Turbulence Model for Aerodynamic Flows", AIAA 92-0439, January 1992.

37. Spalding, D. B. : *Journal of Applied Mechanics*, Vol. 28, pp. 455-457.

38. Smith, William D. : "Improved Pressure and Lift Predictions in Transonic Flow Using in Transonic Flow Using an Unstructured Mesh euler Method with an Interacting Boundary Layer", Masters of Science Thesis, The George Washington University, July 1994.

39. Cebeci, T., Clark, R. W., Chang, K. C., Halsey, N. D., and Lee, K. : "Airfoils with Separation and the Resulting Wakes", *Journal of Fluid Mechanics*, Vol. 163, pp. 323-347, 1986.

40. Lighthill, M. J. : "On Displacement Thickness", *Journal of Fluid Mechanics*, Vol. 4, Part 4, pp. 383-392, 1958.

41. Potsdam, M. A. : "An Unstructured Mesh Euler and Interactive Boundary Layer Method for Complex Configurations", AIAA Paper 94-1844, June 1994.

42. Campbell, Richard L : "An Approach to Constrained Aerodynamic Design With Application to Airfoils", NASA TP 3260, November, 1992.

43. Batina, J. T. : "Unsteady Euler Algorithm with Unstructured Dynamic Mesh for Complex Aircraft Aeroelastic Analysis", AIAA 89-1189, 1989.

44. Parikh, P. and Frink, N. T. : "An Adaptive Remeshing Procedure for three Dimensional Unstructured Grids", *4th International Symposium on Computational Fluid Dynamics*, Davis, CA, September 1991.

45. Parikh, P. : "An Adaptive Remeshing Scheme For Vortex Dominated Flows Using Three Dimensional Unstructured Grids", ICASE/LaRC *Workshop on Adaptive Grid Methods*, Hampton, VA, November 7-9, 1994. (To be published in NASA Conference Proceedings.)

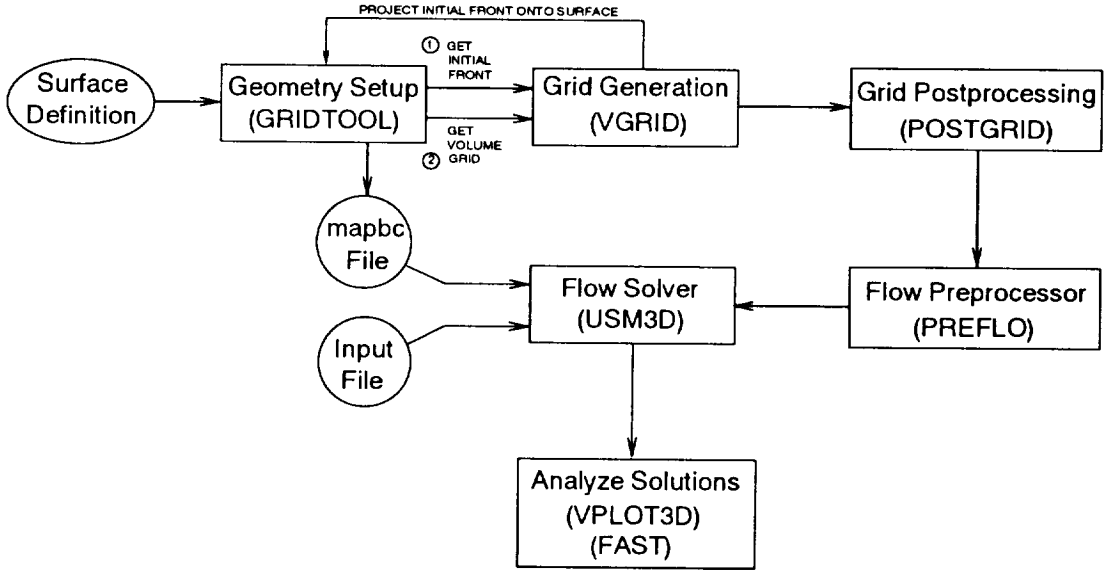



Figure 1. - Flowchart of unstructured-grid flow analysis system.

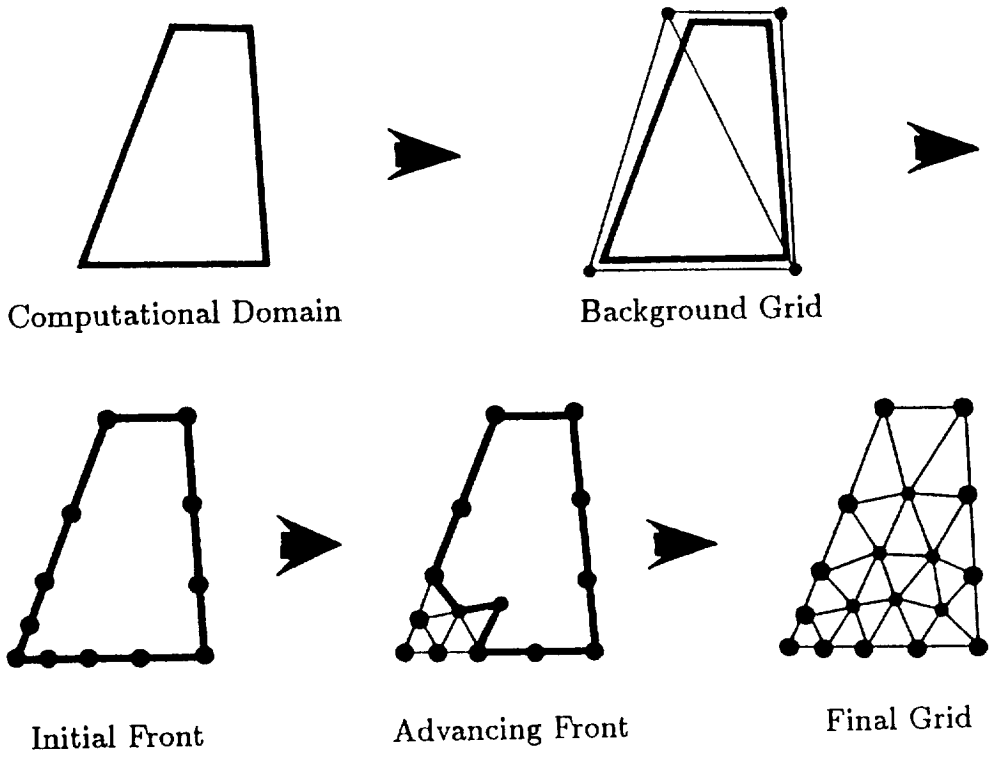



Figure 2. - Advancing-front process for a simple 2D domain

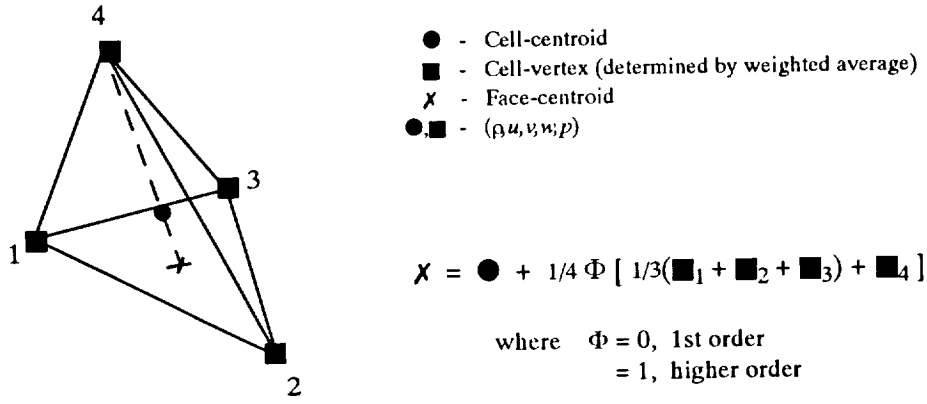



Figure 3. - Higher-order stencil for tetrahedral cell-centered scheme.

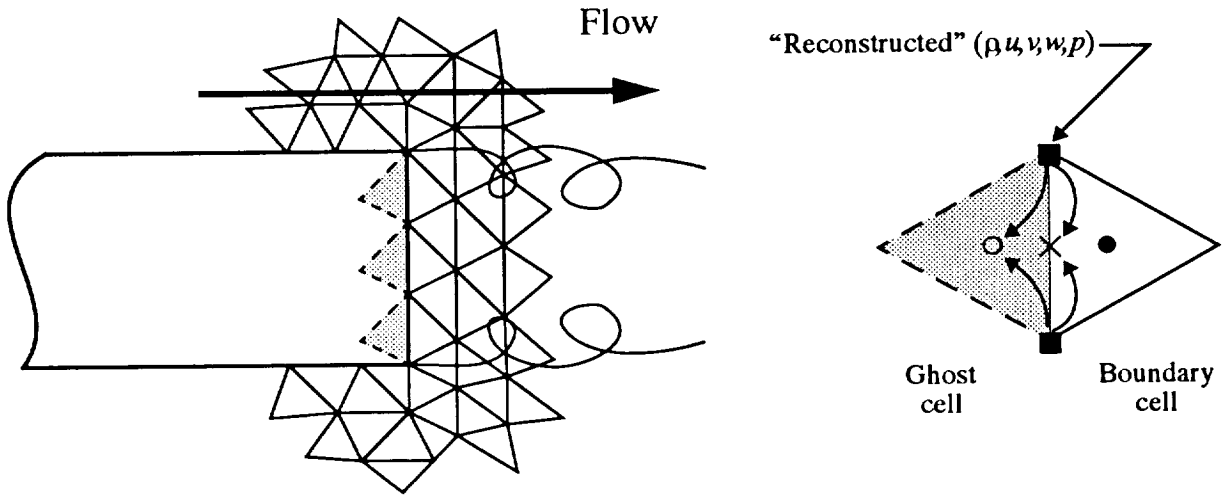



Figure 4. - Schematic of blunt-base boundary condition

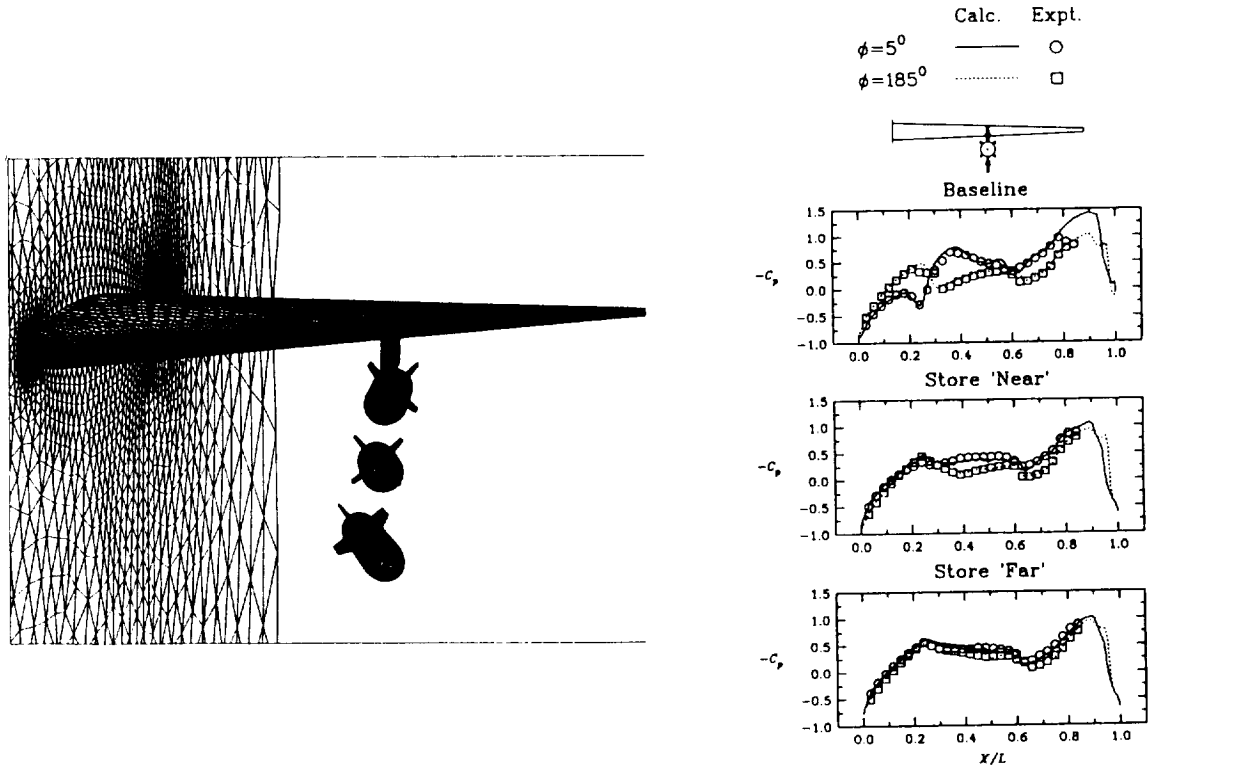



Figure 5. - - Surface triangulation on a wing/pylon/store configuration, and surface pressure comparisons on store along streamwise rays, $M_\infty$=0.95.

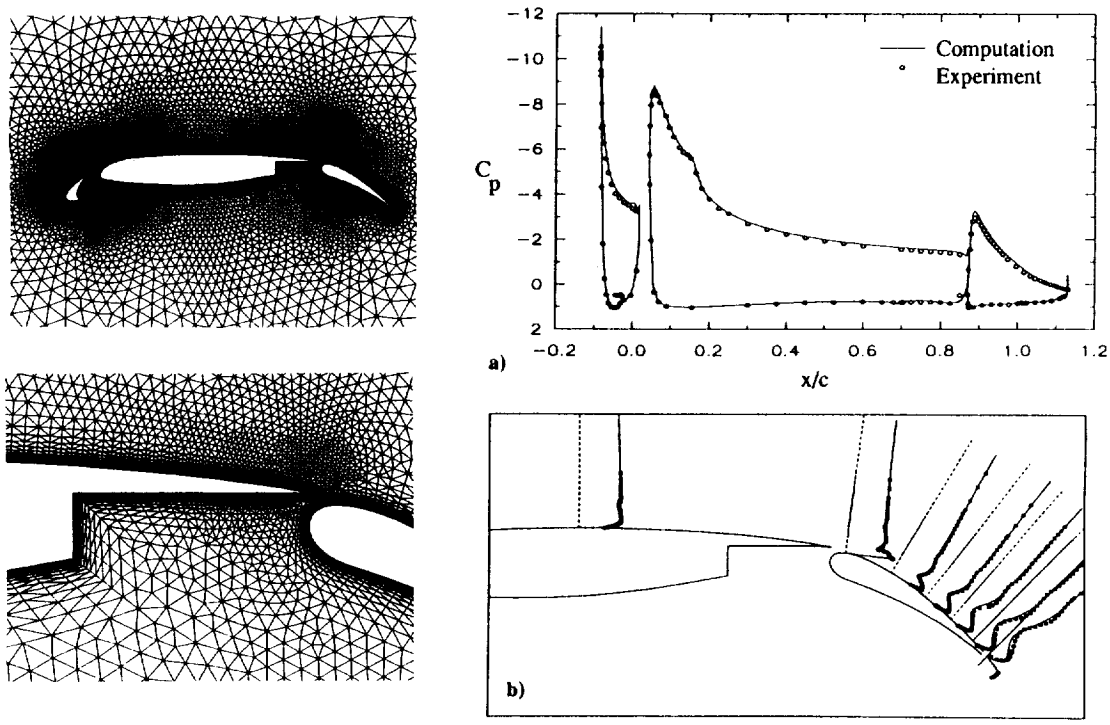



Figure 6. - Unstructured triangular viscous grid and viscous-flow solution for multi-element airfoil at $M_\infty=0.2$, $a=16.2^o$, and $Re_c=9X10^6$: a) surface pressure distributions and b) velocity profiles.

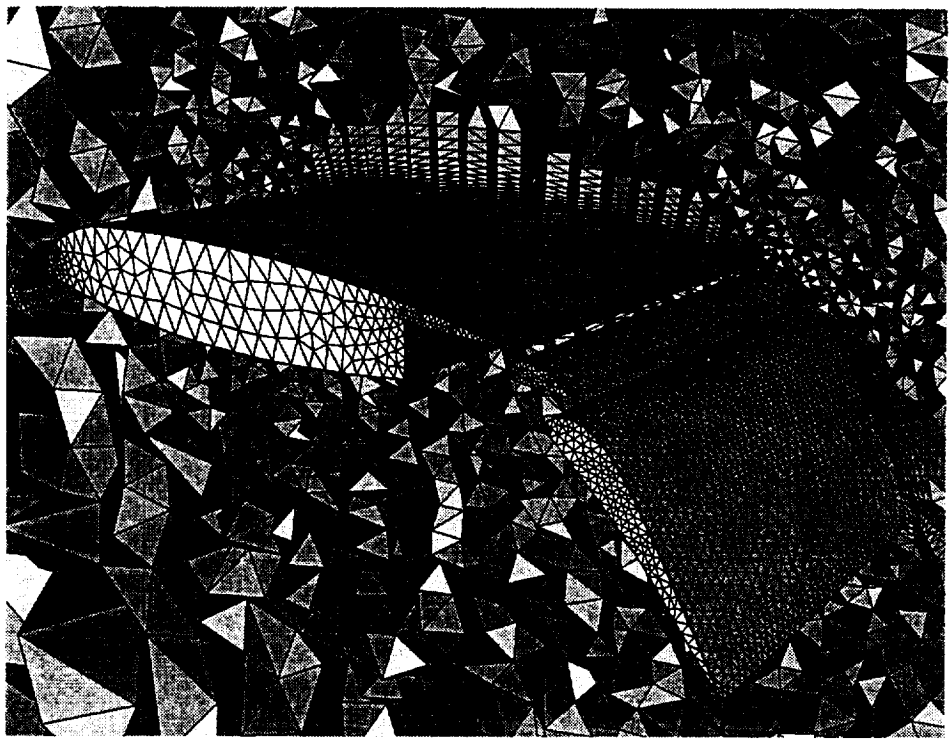
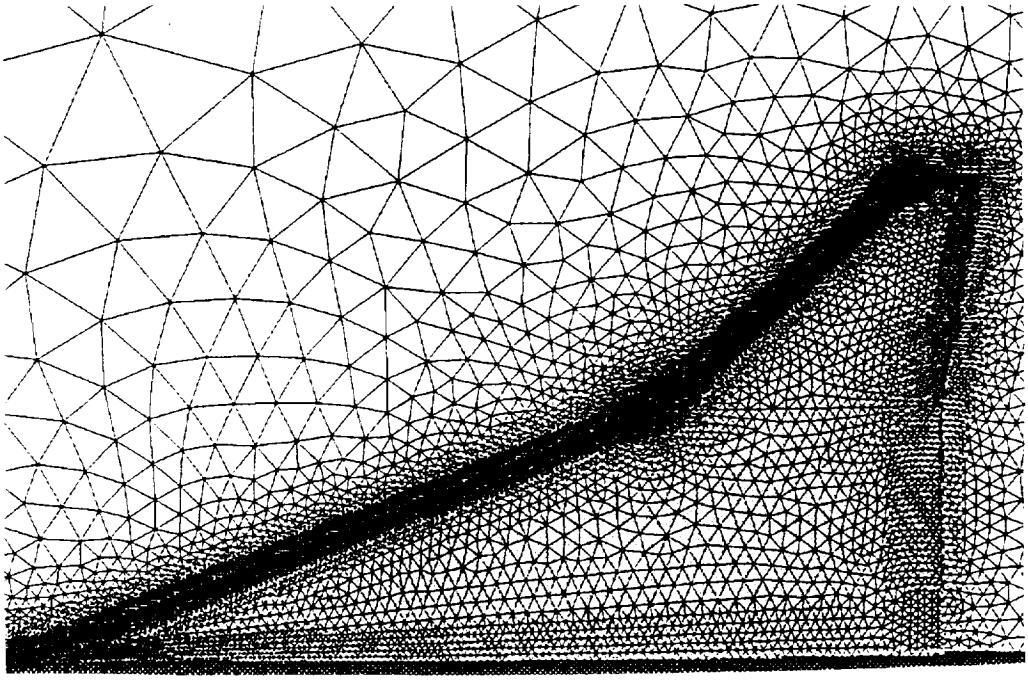
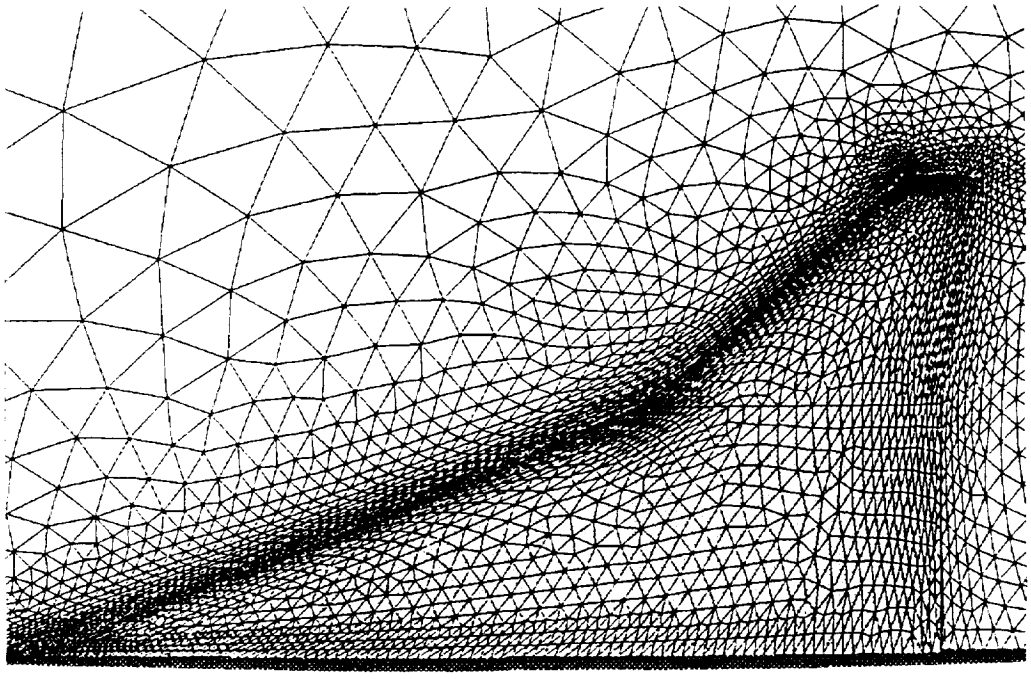


Figure 7. - Complete tetrahedral viscous grid on a multi-element wing shown on a plane.



(a) Isotropic surface and field grid (40,778 triangles on wing).



(b) Multidirectional anisotropically stretched surface and field grid (6,841 triangles on wing).

Figure 8. - Example of multidirectional anisotropic grid stretching for cranked-wing configuration.

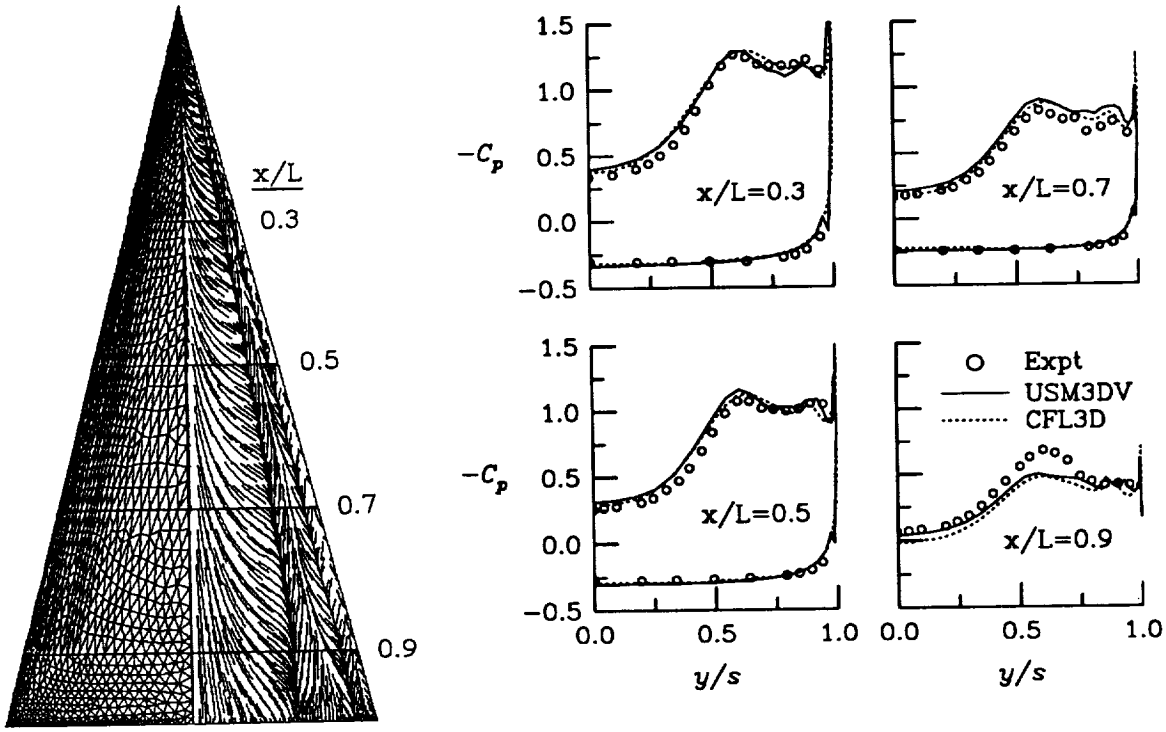



Figure 9. - Laminar flow over aspect-ratio 1 delta wing for $M_\infty$=0.3, $a$=20.5°, $Re_L$=0.9X10$^6$. Unstructured surface grid and "oil-flow" pattern (left), pressure profiles at four chord stations (right).

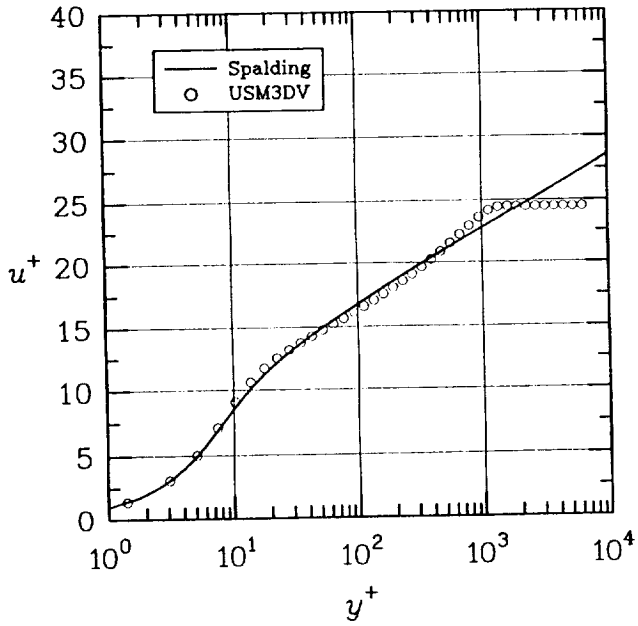



Figure 10. - Comparison of computed law-of-the-wall with analytical model for flat plate boundary layer flow, $M_\infty$=0.5, $Re_L$=2X10$^6$.

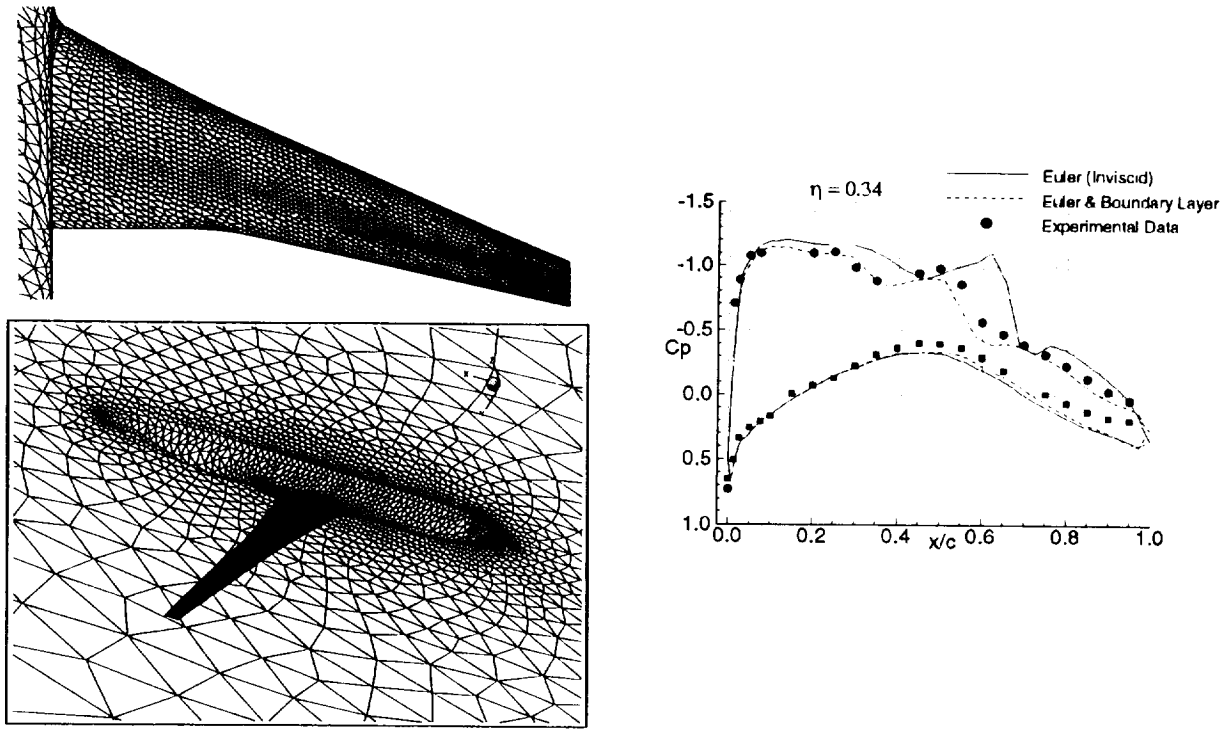



Figure 11. - Effect of interacting boundary layer on pressure predictions for a low-wing transport. $M_\infty$=0.77, $\alpha$=1.12°, and $Re_c$=2.6X10$^6$

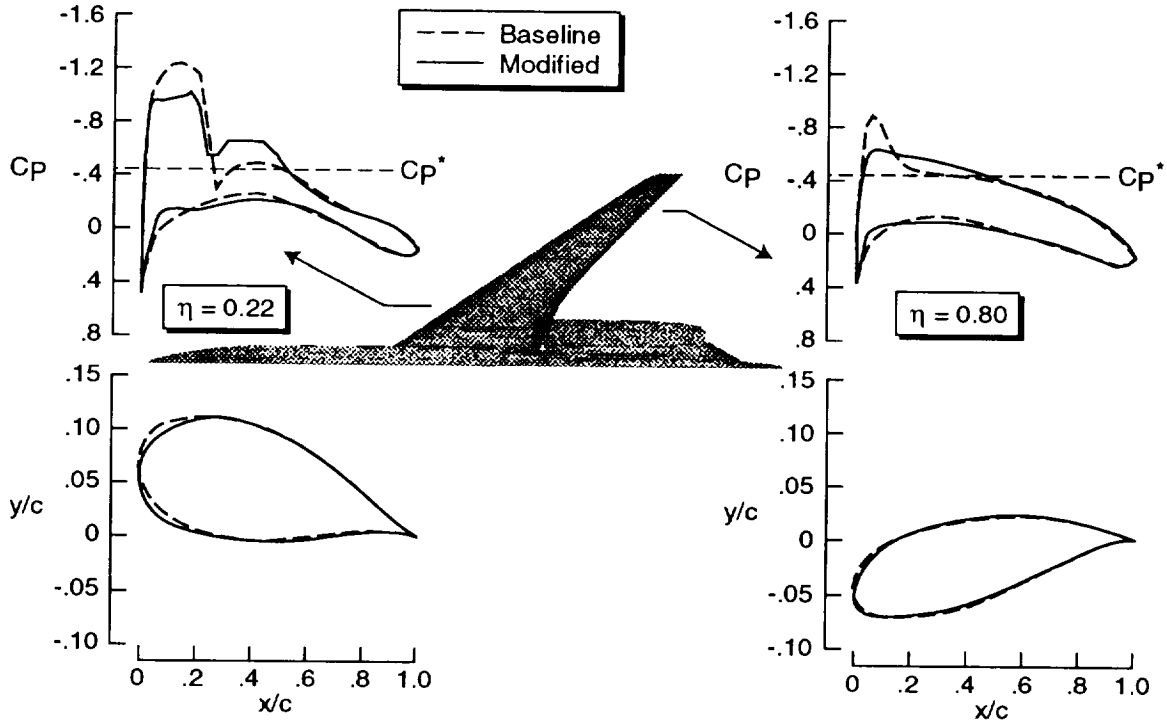



Figure 12. - Application of USM3D/CDISC design method to wing redesign of an executive transport configuration, $M_\infty$=0.80.

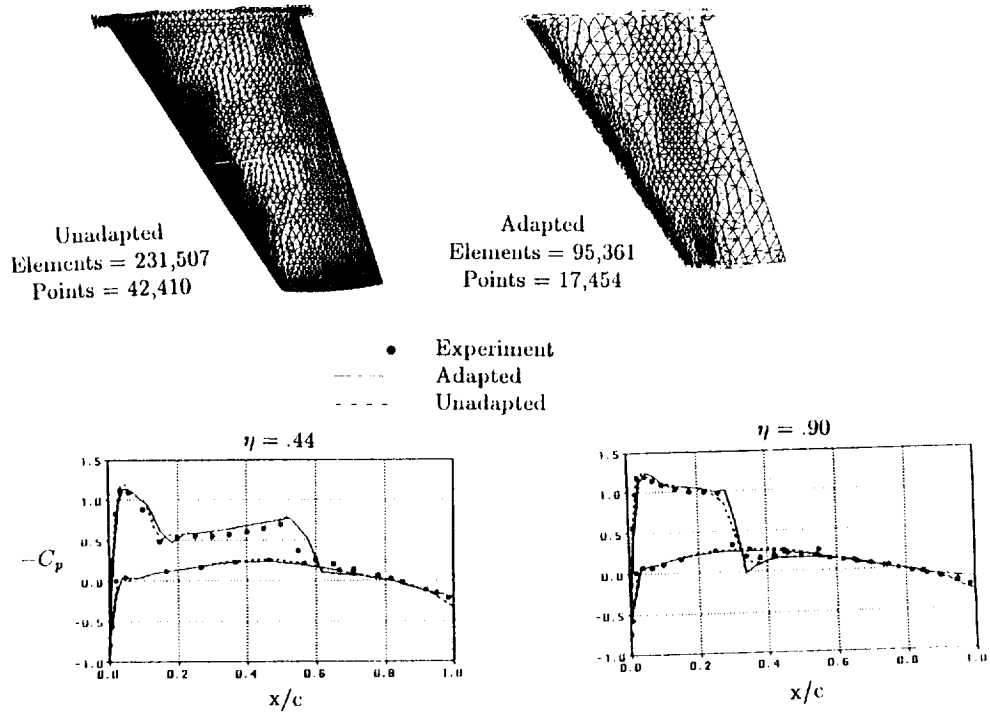



Figure 13. - Solution derived grid adaptation to shock dominated flow features for ONERA M6 wing, $M_\infty=0.84$, $a=3.06^o$.

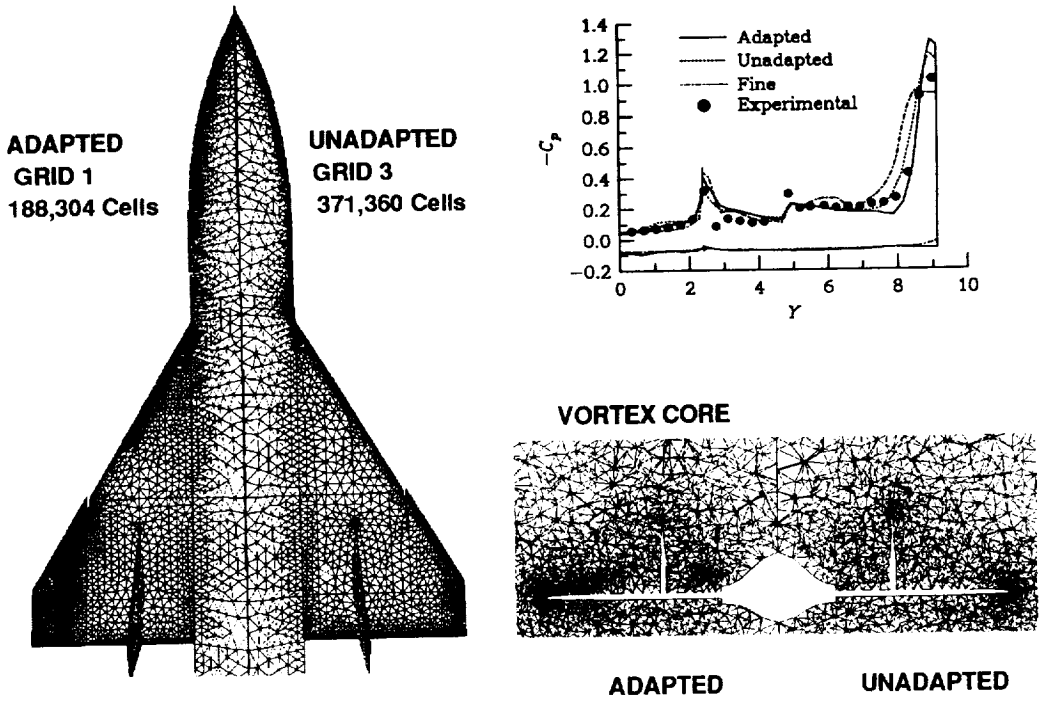



Figure 14. - Solution derived grid adaptation to vortex-flow features for MTVI configuration, $M_\infty=0.4$, $a=10.5^o$. Surface triangulation (left), field adaptation at $x/c_r=0.93$ (lower right), and surface pressure comparison at $x/c_r=0.93$ (upper right).

# HYBRID GRID TECHNOLOGY

N95-28744

# ADAPTIVE HYBRID PRISMATIC-TETRAHEDRAL GRIDS FOR VISCOUS FLOWS

Yannis Kallinderis *, Aly Khawaja†, and Harlan McMorris‡

Dept. of Aerospace Engineering and Engineering Mechanics
The University of Texas at Austin
Austin, TX 78712

## ABSTRACT

The paper presents generation of adaptive hybrid prismatic/tetrahedral grids for complex 3-D geometries including multi-body domains. The prisms cover the region close to each body's surface, while tetrahedra are created elsewhere.

Two developments are presented for hybrid grid generation around complex 3-D geometries. The first is a new octree/advancing front type of method for generation of the tetrahedra of the hybrid mesh. The main feature of the present advancing front tetrahedra generator that is different from previous such methods is that it does not require the creation of a background mesh by the user for the determination of the grid-spacing and stretching parameters. These are determined via an automatically generated octree.

The second development is an Automatic Receding Method (ARM) for treating the narrow gaps in between different bodies in a multiply-connected domain. This method is applied to a two-element wing case.

A hybrid grid adaptation scheme that employs both $h$-refinement and redistribution strategies is developed to provide optimum meshes for viscous flow computations. Grid refinement is a dual adaptation scheme that couples division of tetrahedra, as well as 2-D directional division of prisms.

## INTRODUCTION

Simulation of flows around three-dimensional bodies is a major issue in computational fluid mechanics. Geometrical and flow-field complexity combine to make 3-D computations a pacing item. The generation of a body-conforming grid has proven to be a difficult task [1, 2].

The success of a structured grid generation may be extremely dependent on geometry and operator proficiency. Block-structured schemes exist which, based on extensive user input, break the computational domain into a number of blocks within which hexahedra are constructed. A radical alternative to structured meshes is to use tetrahedra. Tetrahedral grids provide flexibility in 3-D grid generation since they can cover complicated topologies easier compared to the hexahedral meshes. This does not come without a price, viz., unstructured grids require a great deal more memory than their structured counterparts. They

*Associate Professor
†Graduate Research Assistant
‡Graduate Research Assistant



PRECEDING PAGE BLANK NOT FILMED

employ pointers to provide connectivity information between cells, faces, edges, and nodes. Additionally, approximately five to six times more tetrahedra than hexahedra are required to fill a given region with a fixed number of nodes. This vast increase in the number of required cells leads directly to impractical memory requirements for 3-D viscous flow simulations. Furthermore, resolution of the strong directional flow gradients encountered in viscous flows requires very thin grid elements. It is very expensive to generate tetrahedral cells with high aspect ratio to resolve such gradients.

One solution to the dilemma between hexahedra and tetrahedra is to use a semi-unstructured grid made of prisms. Prismatic cells are composed of triangular faces in the lateral (body-surface) directions and quadrilateral faces in the normal direction. Therefore, they can provide the geometric flexibility of unstructured grids as well as the orthogonality and high aspect ratio qualities of structured grids. Results have been obtained using prismatic grids that reveal their suitability for resolving viscous flow phenomena [3, 4, 5]. The prismatic grid requires a set of pointers to define their base triangular mesh combined with a single index for each prism belonging to the same stack [4]. Finally, structure of the prisms in one of the directions can be exploited in order to apply directional multigrid acceleration within the viscous regions [4, 6].

The areas between different prismatic layers covering the surfaces of the domain can be quite irregular. Furthermore, the relevant flow features do not usually exhibit the strong directionality that the viscous stresses have. Tetrahedral elements appear to be appropriate for these irregularly shaped regions. Their triangular faces can match the corresponding triangular faces of the prisms.

The present work employs two families of grid elements: prismatic grid cells for the viscous region and tetrahedral grid cells elsewhere. A new advancing front type of method is developed for generation of the tetrahedra of the hybrid (prismatic/tetrahedral) mesh. The main feature that is different from previous advancing front generators [7, 8] is that it does not require a user-constructed background mesh for determination of the grid-spacing and stretching parameters. It should be noted that generation of the background mesh has been a very time-consuming and user-dependent part of previous advancing front methods. A special octree is constructed via a *Divide-and-Conquer* method of the space outside of the region covered by the prisms. The grid spacing is then determined based on the size of local octants which form the octree.

An important issue arising with use of semi-structured prismatic grids is covering of narrow gaps in between different bodies. In the present work, an Automatic Receding Method (ARM) is developed that relies on receding the prisms layers that surround each body. The gap that the prisms leave is filled with tetrahedra. The case of a two-element wing is considered as a test of the technique.

A High Speed Civil Transport (HSCT) type of aircraft geometry is considered in order to investigate efficiency and to demonstrate robustness of the method in handling relatively complex topologies. The generated hybrid grid required only 170 K tetrahedra instead of an estimated two million had a tetrahedral mesh been used in the prisms region, as well.

A hybrid grid adaptation scheme that employs both $h$-refinement and redistribution strategies is developed to provide optimum meshes for viscous flow computations. Grid refinement is a dual adaptation scheme that couples division of tetrahedra, as well as 2-D directional division of prisms.

## GENERATION OF PRISMS

An unstructured triangular grid is employed as the starting surface to generate a prismatic mesh. This grid, covering the body surface, is marched away from the body in distinct steps, resulting in generation of semi-structured prismatic layers in the marching direction. The goal of the marching scheme is to reduce

the curvature of the previous marching surface at each step while ensuring smooth grid spacing to avoid surface overlap. The process can be visualized as a gradual inflation of the body's volume. There are three main aspects of the algebraic grid generation process: (i) determination of the directions along which the nodes will march (marching vectors), (ii) determination of the distance by which the nodes will march along the marching vectors, and (iii) smoothing operations on positioning of the nodes on the new layer.

### Determination of the Marching Vectors

Each node on the marching surface is advanced along a marching vector. The marching direction is based on the *node-manifold*, which consists of the group of faces surrounding the node to be marched. The primary criterion to be satisfied when marching is that the new node should be visible from all the faces on the manifold.

The node-normal vector lies on the bisection plane of the two faces on the manifold that form the wedge with the smallest angle. Its location on this plane is determined by bisecting the visibility region on the plane. This process has yielded consistently valid normal vectors at the nodes by constructing the vector most normal to the most acute face planes. Essentially, it does this by maximizing the minimum angle between the node-normal and all the surrounding face normals. A more detailed description of the marching procedure can be found in [3].

Since the visibility requirement is a necessary one in order to obtain a valid grid, all subsequent smoothing operations performed on the original normal vector enforce the visibility constraint.

### Marching Step Size

Determination of marching distances is based on the characteristic angle $\beta_{ave}$ of the manifold of each node to be marched. This angle is computed using the average dot product between the pairs of faces forming the manifold. The marching distance is a linear function of $\beta_{ave}$. It yields relatively large marching distances in the concave regions, and small distances in the convex areas of the marching surface. Specifically, the distance $\Delta n$ is:

$$\Delta n = (1 + \alpha)\Delta n_{ave}, \tag{1}$$

where $\Delta n_{ave}$ is the average marching step for that layer, and $\alpha$ is a linear function of the manifold angle $\beta_{ave}$. The sign of $\alpha$ is positive for concave regions and negative for convex regions.

The average marching step for each layer, $\Delta n_{ave}$ is computed based on a user specified initial marching step $\Delta n_o$ and a stretching factor $st$. The actual marching step for layer $j$ is given by:

$$\Delta n^j_{ave} = \Delta n_o \times st^{(j-1)}. \tag{2}$$

### Smoothing Steps

The initial marching vectors are the normal vectors. However, this may not provide a valid grid since overlapping may occur especially in regions of the grid with closely spaced nodes. To prevent overlapping, the directions of the marching vectors must be altered. A number of smoothing passes (typically 5) are performed over all the nodes on the marching surface. Weighted Laplacian smoothing is applied to the direction of the marching vectors as follows:

$$\vec{V}_i = \omega \vec{V}'_i + \frac{(1-\omega)}{\sum_j 1/d_{ij}} \sum_j (1/d_{ij})\vec{V}_j, \tag{3}$$

where $\vec{V}_i'$ and $\vec{V}_i$ are the initial and final marching vectors of node $i$ , while $\vec{V}_j$ are the marching vectors of the surrounding nodes $j$ that belong to the manifold of node $i$. The weighting factor $\omega$ is a function of the manifold characteristic angle $\beta_{ave}$. It has small values in concave regions, and relatively large ones in convex areas. The averaging of the marching vectors of the neighboring nodes is distance-weighted with $d_{ij}$ denoting the distance between nodes $i$ and $j$.

A similar procedure is employed for the smoothing of the marching steps $\Delta n$.

### Constraints Imposed to Enhance Quality

Typical Navier-Stokes integration methods impose restrictions on the spacing of the points along the marching lines, as well as on smoothness of these lines. In other words, the prismatic grid should not be excessively stretched or skewed. In the present work, two constraints are imposed on the positioning of points on the marching lines, as well as on the deviation of the direction of the marching vectors from one layer to the next. The ratio of the grid spacings along the marching lines of any two consecutive prisms layers should be less than a stretching factor $st$:

$$(1 - st)\Delta n|_{j-1} < \Delta n|_j < (1 + st)\Delta n|_{j-1}, \tag{4}$$

where the subscripts $j - 1$ and $j$ denote the order of points along each marching line. A typical value of the allowed stretching factor $st$ is 0.2. Furthermore, the angle between two consecutive marching vectors $\vec{V}_{j-1}$ and $\vec{V}_j$ should be less than a specified angle of 30°.

### Scalability of Prisms Generation Time

Grid generation time depends on the number of boundary faces, as well as on the number of prisms layers that are created. The cases of mesh generation around an ONERA M6 wing and an HSCT configuration are employed in order to study how the time scales with these two factors. Three different surfaces triangulations of the M6 wing are considered. The first consists of 3239 faces, the second of 8807 faces, while the third has 15279 faces. Figure 1 shows almost linear increase in generation time with number of boundary faces. The number of generated prisms layers was kept the same in all three cases and equal to 24. Perfect linear scaling of the mesh generation time with number of prisms layers is demonstrated in Figure 2. This implies that the required operations of the generator are exactly the same for each layer.

## OCTREE/ADVANCING FRONT TETRAHEDRA GENERATION

A new method is presented for generating the tetrahedra of the hybrid grid. Advancing front type of methods require specification by the user of the distribution of three parameters over the entire domain to be gridded. These field functions are: (i) the node spacing, (ii) the grid stretching, and (iii) the direction of the stretching. In the present work these parameters do not need to be specified. The distribution of grid size, stretching, and direction of stretching is automatically determined via an octree. There is no need for a special background mesh which has been the backbone of previous advancing front generators.

The tetrahedra that are generated should progressively become larger as the front advances away from the original surface. Their size, the rate of increase of their size, as well as the direction of the increase are given from an octree consisting of cubes which is generated automatically via a *Divide-and-Conquer* method. This process generates octants that are progressively larger with distance away from the body. Their size will be the characteristic size of the tetrahedra that will be generated in their vicinity.

Generation starts from the outermost surface of the layer of prisms surrounding the body. The triangles of this surface form the initial front. Then, a list of points is created that consists of a new node, as well as of "nearby" existing points of the front. One of these points is chosen to connect to the vertices of the face. Following choice of the point to connect to, a new tetrahedron is formed. The list of the faces, edges, and points of the front is updated by adding and/or removing elements. The algorithm followed in the present work is the one presented in [8, 9]. The method requires a data structure which allows for efficient addition/removal of faces, edges and points, as well as for fast identification of faces and edges that intersect a certain region. The alternating digital tree (ADT) algorithm is employed for these tasks.

## A Special Octree for Tetrahedral Grid Spacing Control

The *divide-and-conquer* process starts with a *master* hexahedron that contains the body. This hexahedron is recursively subdivided into eight smaller hexahedra called *octants*. Any octant that intersects the body is a *boundary octant* and is subdivided further (inward refinement). The subdivision of those boundary octants ceases when the size of the boundary octant matches the thickness of the prisms on the outermost prisms surface.

Then, the hexahedral grid is further refined in a *balancing process* (outward refinement) to prevent neighboring octants whose depth differs by more than one (*interface octants*). Outward refinement is performed to ensure that the final octree varies smoothly in size within the areas not covered by the prisms. The sole criterion for outward refinement is a depth difference greater than one between the octant itself and any of its neighbors. Only interface octants are subject to outward refinement. Figure 3 illustrates the growth of the size of the octants away from the surface. The figure shows the outermost prisms surface around a High Speed Civil Transport (HSCT) type of aircraft geometry, as well as a cut through the octree. Growth of the octants away from the outermost prisms surface guides growth of the corresponding tetrahedra. Figure 4 illustrates the symmetry plane of the HSCT geometry. The quadrilaterals (dark lines) correspond to the faces of the octants on this plane, while the triangles (light lines) correspond to the faces of the tetrahedra. It is observed that the size of the tetrahedra, as well the stretching of the mesh and the direction of stretching is guided quite accurately by the octree.

Simplicity and no user intervention are main advantages of the octree. The usual trial-and-error procedures for constructing the field functions that give the local size of the tetrahedra, the stretching of the mesh, and the direction of the stretching (background mesh) for previous advancing front generators are avoided in the present method. The octree is generated once and remains the same throughout the generation process.

## Determination of Size of the Tetrahedra

The advancing front method creates a new tetrahedron by connecting each face of the current front to either a new or an existing node. This point is found by using a characteristic distance $\delta$ which is calculated from the size of the local octant to which the face of the front belongs.

The local characteristic size $\delta$ is calculated as follows:

$$\delta = H s t^l, \tag{5}$$

where $H$ is the size of the boundary octants, $st$ is the stretching parameter, and $l$ is the level of the local octant to which the face of the front belongs. The values of $l$ range from 0 (boundary octant) up to a number equal to the number of recursive subdivisions of the initial (master) hexahedron. A typical value

of the stretching parameter $st$ is 1.8. The smaller the value of $st$, the smoother the variation in size of the generated tetrahedra. However, a very small value results in generation of a very large number of elements. The chosen value is a compromise between the two effects. Further details of the method are given in [11].

## THE AUTOMATIC RECEDING METHOD (ARM) FOR HYBRID GRID GENERATION AROUND MULTI-BODY DOMAINS

The developed hybrid grid generation method is flexible and general in order to treat domains that contain multiple bodies. A prismatic layer is created around each one of the bodies, while the regions in between these meshes are filled with tetrahedra. Any location and orientation of these bodies is allowed. This is accomplished via a special method for treatment of narrow gaps that frequently form in multiply-connected domains, such as multi-element wings. The key feature of the method is the fact that the prismatic grid around each of the bodies is generated independently of all the other bodies. As a result, such generation is as simple as the generation of prisms for a domain containing a single body. However, overlapping meshes are avoided here by employing a special technique that redistributes the prisms nodes along their corresponding marching lines after the initial generation. This redistribution occurs in the vicinity of the regions of overlapping prismatic meshes and results in formation of gaps in between the previously overlapping prisms layers. Then a tetrahedral grid is generated in order to fill in those gaps. It should be emphasized that the structure of the prismatic grid is not destroyed.

### Receding of the Prism Layers

Receding of the prisms nodes occurs along the marching lines that intersect with another prismatic mesh. The distance over which the outermost point is "pulled-back" depends on the local extent of overlapping. In order to avoid abrupt changes in the thickness of the layer due to the local receding, the nodes belonging to the neighboring marching lines are also receded to a certain extent. This extent gradually reduces to zero away from the area of overlapping of the meshes. Furthermore, the marching line is not altered due to the rearrangement of the points on it. Finally, redistribution of the points along each marching line obeys the constraint on allowable stretching of the mesh. As a result, the spacing of the first point off the surface is reduced which will have no adverse effect on accuracy of solutions.

The steps that are followed in order to remove overlapping of a specific pair of prism layers are:

1. Find the marching lines of each one of the separate prismatic meshes that intersect with one or more of the other grids.

2. Calculate the length of overlapping of each one of the marching lines of the two or more prismatic meshes that overlap.

3. Redistribute the nodes on these marching lines so that no overlapping occurs. Essentially, this results in receding of the nodes closer to the body surface.

4. Avoid abrupt changes in the thickness of the prisms layers by receding neighboring marching lines that do not intersect. This is accomplished by flagging the neighboring marching lines and calculating the distance of receding ($\Delta n^r$) according to:

$$\Delta n^r = \frac{\sum_{neib}(\Delta n^r)_{neib}}{N} \tag{6}$$

where the subscript *neib* denotes the marching lines that are neighboring the specific line, and $N$ is the number of neighboring lines. This lateral smoothing process is repeated a few times (typically 10).

An integral part of the previous method of receding is repositioning of all the points on the marching lines that are 'pulled-back'. A scheme is employed which redistributes the nodes so that the shape of the lines is not altered. In other words, the marching directions are maintained, but the marching distances between consecutive points are modified. This is accomplished by performing a cubic-spline fit to each of the marching lines using the prism node locations for the spline knots. The nodes are then redistributed along the splined lines. The distribution is such that the new node-positions satisfy the grid spacing constraint. In the present work, a certain stretching factor is maintained, while the spacing of the first point off the body surface is reduced.

### Application to Two-Element Wing with Variable Gap Size

In order to illustrate validity of the previous procedure, the case of a two-element wing with variable size of the gap between the main wing and its flap is considered. Figure 5 shows the geometry of the two-element wing. The gap increases along the span. Stage one involves generation of the two separate prismatic meshes that cover each one of the two bodies. Generation is quite simple due to the fact that each layer of prisms is grown independently of the other layer. The two grids overlap locally as shown in Figure 6. In the second stage, the thickness of the prisms layers is reduced locally and the overlap no longer occurs as shown in Figure 7. Comparing the grids of Figures 6 and 7, it is observed that the receding occurs over a larger region which results in a smooth variation of the local thicknesses of both meshes. The final stage involves generation of the tetrahedral mesh that covers all areas in between the prisms. Figure 8 shows the final hybrid (prismatic/tetrahedral) grid on the plane of symmetry. The quadrilaterals are the signature of the prisms on that plane, while the triangles correspond to faces of the tetrahedral mesh.

## HYBRID GRID GENERATION AROUND THE HSCT

A High Speed Civil Transport (HSCT)-type of aircraft geometry was chosen as the test case for the developed grid generator. Figure 9 shows the triangulation of the initial surface. The mesh consists of 4412 triangles and 2275 nodes. A symmetry plane is considered that divides the body. Thus, hybrid grid is generated for half of the space.

The time required to generate the prismatic grid around the HSCT was 90 seconds for 40 layers of prisms on an IBM 390 workstation. Generation of approximately 170,000 tetrahedra took about 67 minutes on the same station. It should be emphasized that employment of a hybrid grid for the HSCT geometry required only 170 K tetrahedra instead of an estimated two million had a tetrahedral mesh been used in the prisms region, as well.

A view of the grown prismatic surface is shown in Figure 10. The growth of the grid is illustrated after 40 marching steps. The effect of the marching process is similar to inflating of the original body volume. Figure 11 shows portion of the initial wing surface, as well as the outer surface of the prisms. Both the structured part (quadrilateral faces), and the unstructured part (triangular faces) of the prisms is shown. Every fourth layer of prisms is shown for clarity of the plot. Two wire-frame views of the prismatic grid are illustrated in Figure 12. The shaded regions correspond to the surface of the aircraft.

Three different stages in the growth of the tetrahedral mesh on top of the prisms is shown in the sequence of Figure 13. The space between the outer prisms surface and the farfield is 'filled up' quite fast.

It is worth observing the transition of the hybrid mesh from the prisms to the tetrahedra. Figure 14 shows the hybrid grid at a section of the symmetry plane of the aircraft. The quadrilateral faces correspond to the prisms, while the triangular faces belong to the tetrahedra. It is observed that the grid transitions smoothly from the prisms region to the tetrahedral area.

The final hybrid grid consists of 176,480 prisms and of 170,300 tetrahedra. Figure 15 shows the signature of the mesh on the symmetry plane. Finally, Figure 16 illustrates the hybrid mesh on two different planes that are perpendicular to each other. The first plane is the symmetry and it is indicated by the darker lines, while the second is intersecting the fuselage at a location upstream of the wing and it is shown via light lines. It should be noted that the irregularity of the lines observed on the second plane are due to the fact that the grid it intersects is not planar as it is on the symmetry plane.

## COMBINED REFINEMENT/REDISTRIBUTION FOR HYBRID GRIDS

A dynamic grid adaptation algorithm has previously been developed for 3-D unstructured grids [10]. The algorithm is capable of simultaneously refining and un-refining appropriate regions of the flow domain. This method is extended to the present work and is coupled with prismatic grid adaptation to implement a hybrid grid adaptation method.

### Directional Division of Prisms

The prisms are refined directionally in order to preserve the structure of the mesh along the normal-to-surface direction. The prismatic grid refinement proceeds by dividing only the *lateral* edges that lie on the wall surface and hence the wall faces. The faces are divided either into two or four subfaces. The resulting surface triangulation is replicated in each successive layer of the prismatic grid. This results in all the prisms that belong to the same stack (namely, the group of cells that originate from the same triangular face on the wall surface) getting divided alike. The prismatic grid refinement preserves the structure of the initial grid in the direction normal to the surface. The primary advantage of using such an adaptive algorithm for prisms is that the data structures needed for its implementation are essentially as simple as that for refining a 2-D triangular grid.

The directional division of the prisms does not increase resolution of flow features that are aligned in a direction that is normal to the wall surface. However, a grid redistribution algorithm can be employed in order to recluster nodes in the normal direction so as to better resolve the viscous stresses [3, 11].

The tetrahedral cells constitute the portion of the grid where inviscid flow features are dominant. These features do not exhibit the directionality that is generally prevalent in viscous stresses. Hence, the tetrahedra are refined by division into eight, four, or two subcells [10].

### Redistribution of Prisms

The redistribution algorithm increases local grid resolution by clustering existing grid points in regions of interest. A measure of the grid resolution required normal to the *no-slip* wall is the values of $y^+ = \frac{u_\tau y}{\nu}$, with $u_\tau = \sqrt{\frac{|\tau_{wall}|}{\rho_{wall}}}$ being the wall friction velocity. A criterion based on the values of $y^+$ at the wall is employed to either *attract* nodes towards the wall or *repel* them away from the surface so that a specific value of $y^+$ is attained at all the wall nodes. This procedure in essence determines a new value for the spacing $\delta_{wall}$ of the first node off the wall at all locations on the wall surface. The nodes in the prismatic region

are then reclustered along the marching lines emanating from the corresponding wall node, in accordance with the new value of $\delta_{wall}$. Details are presented in [4].

## Application of Hybrid-Adaptation Method

The test case of flow past a sphere at a free stream Mach number of $M_\infty = 1.4$ and a Reynolds number of $Re = 1000$ (based on the radius of sphere) is considered. The flow is characterized by both inviscid and viscous flow features such as shock waves and boundary layer separation. Details are given in [4].

The hybrid grid adaptation algorithm is implemented to obtain a numerical solution for the flow situation discussed above. A coarse hybrid grid comprising $\sim$ *1400* wall boundary nodes and $\sim$ *100K* tetrahedra is used as the initial grid. The prismatic region is constituted by *20* layers of prisms. The locally adapted grid obtained after $h$-refinement based on an initial solution and the Mach number contour lines of the final solution obtained on the adapted grid are shown simultaneously in Figure 17. The figure shows the embedded tessellations on the wall surface, symmetry plane as well as on an equatorial plane cutting through the interior of the grid, normal to the symmetry plane. Mach number contour lines are shown superimposed on the embedded mesh on the equatorial plane in the figure. It is clearly seen that embedding in the tetrahedral region is focussed near the shock location just outside of the prismatic-tetrahedral interface. The prismatic region is also directionally refined near the upstream and downstream sections of the body. This is due to the flow upstream accelerating rapidly from the upstream stagnation point and the flow downstream separating that causes flow gradients in the lateral directions that are detected by the directional adaptive algorithm. The embedded hybrid grid comprises $\sim$ *2500* wall boundary nodes and $\sim$ *275K* tetrahedra. The numerical solution obtained by the solution-adaptive approach is compared with that of the solution obtained on a globally refined grid, starting with the same initial coarse mesh as before. The values of skin-friction coefficients computed on the wall surface are compared between the two solutions, as shown in Figure 18. The figure shows the excellent agreement of the results computed on the locally adapted grid with that of the globally refined grid.

## SUMMARY

Generation of the tetrahedra was made simpler due to the fact that a background mesh was not required for determination of the mesh spacings. The spacing parameters provided by the octree yielded tetrahedral elements that varied in size smoothly. Furthermore, the octree enabled a smooth transition of the grid from the prisms to the tetrahedra.

The time required to generate the prismatic grid around the HSCT type of configuration was 90 seconds for 40 layers of prisms on an IBM 390 workstation. This time scales almost linearly with the number of boundary faces. Generation of approximately 170,000 tetrahedra took about 67 minutes on the same station.

Employment of a hybrid grid for an HSCT-type of geometry required only 170 K tetrahedra instead of an estimated two million had a tetrahedral mesh been used in the prisms region, as well.

The developed method of receding prisms layers (ARM) for narrow gaps between different bodies was validated via a case of a two-element wing with variable size of the gap between the two elements.

## ACKNOWLEDGMENTS

This work was supported by NASA Grant NAG1-1459 and NSF Grant ASC-9357677 (NYI Program). The authors wish to thank Ms. Karen Bibb of NASA Langley for giving us access to the FELISA mesh generation code.

## References

[1] J. F. Thompson, and N. Weatherill, "Aspects of Numerical Grid Generation: Current Science and Art", AIAA Paper 93-3539-CP, 1993.

[2] T. J. Baker, "Developments and Trends in Three Dimensional Mesh Generation," *Applied Numerical Mathematics*, Vol. 5, pp. 275-304, 1989.

[3] Y. Kallinderis and S. Ward, "Prismatic Grid Generation for 3-D Complex Geometries", *Journal of the American Institute of Aeronautics and Astronautics*, Vol. 31, No. 10, pp. 1850-1856, October 1993.

[4] V. Parthasarathy, Y. Kallinderis, and K. Nakajima, "A Hybrid Adaptation Method and Directional Viscous Multigrid with Prismatic-Tetrahedral Meshes," AIAA Paper 95-0670, Reno, NV, January 1995.

[5] K. Nakahashi, "Optimum Spacing Control of the Marching Grid Generation", AIAA paper 91-0103 , 1991.

[6] Parthasarathy, V. and Kallinderis, Y., "Directional Viscous Multigrid Method on Adaptive Prismatic Meshes ," *AIAA Journal,* Vol. 33, No. 1, January 1995.

[7] R. Lohner, and P. Parikh, "Generation of Three-Dimensional Unstructured Grids by the Advancing-Front Method ," AIAA Paper 88-0515, 1988.

[8] J. Peraire, J. Peiro, L. Formaggia, K. Morgan and O.C. Zienkiewicz, "Finite Element Euler Computations in Three Dimensions," *AIAA 26th Aerospace Sciences Meeting*, Reno, AIAA Paper 88-0032, January 1988

[9] J. Peiro, J. Peraire, and K. Morgan, "FELISA System Reference Manual", 1994.

[10] Y. Kallinderis and P. Vijayan, "An Adaptive Refinement Coarsening Scheme for 3-D Unstructured Meshes", AIAA Journal, Vol 31, No.8, pp 1440-1447, Aug. 1993.

[11] Y. Kallinderis, A. Khawaja, and H. McMorris "Hybrid Prismatic / Tetrahedral Grid Generation for Complex Geometries", AIAA Paper 95-0211, Reno, NV, January 1995.

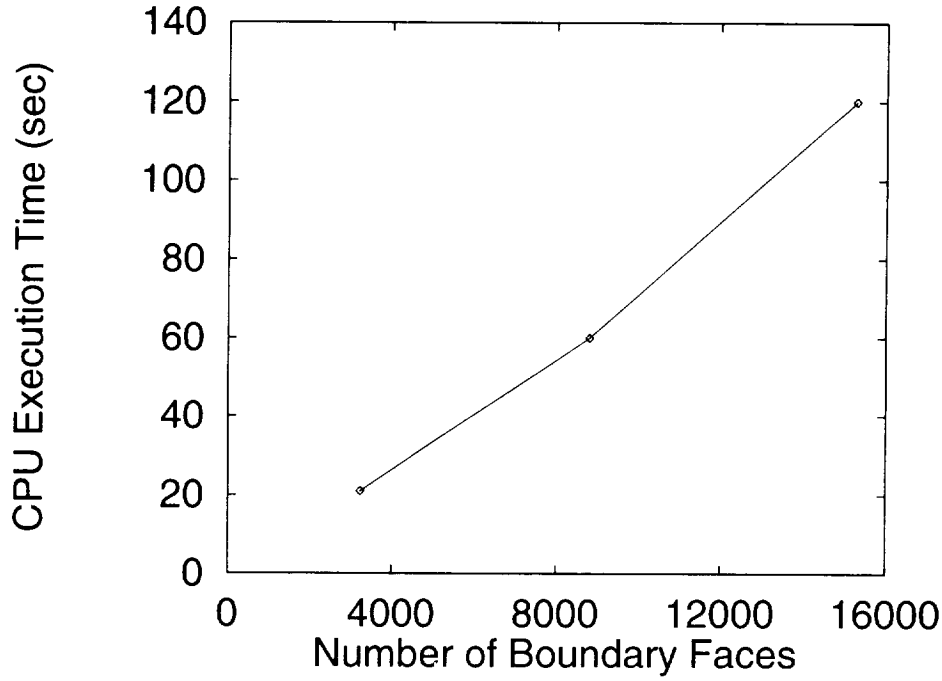



Figure 1: Scaling of prisms generation time with number of boundary faces.
Case of the ONERA M6 wing with 24 layers of prisms.

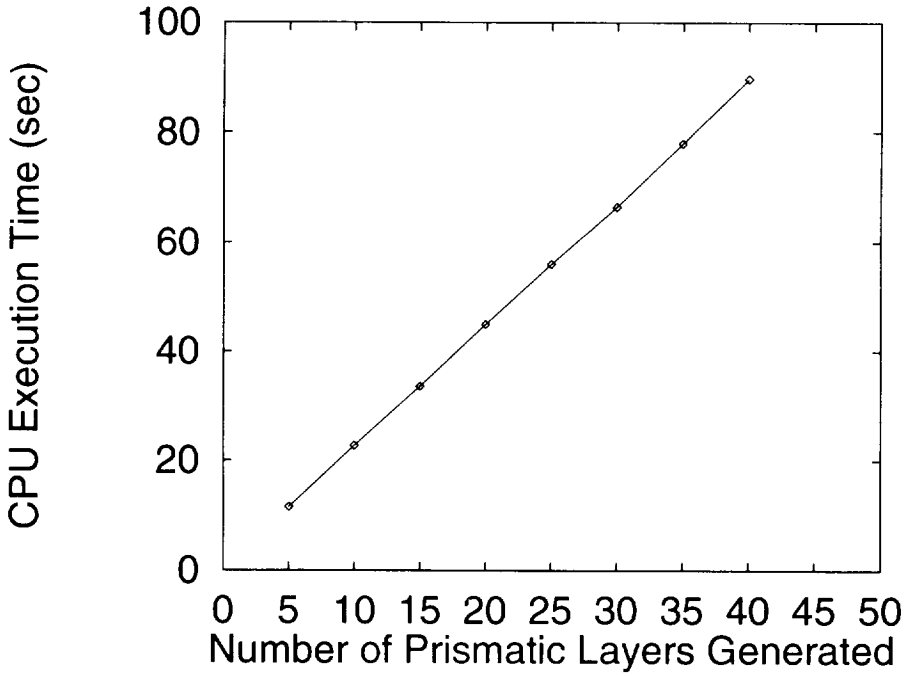



Figure 2: Linear increase in prisms generation time with number of prismatic layers.
Case of the HSCT aircraft with 4412 boundary faces.

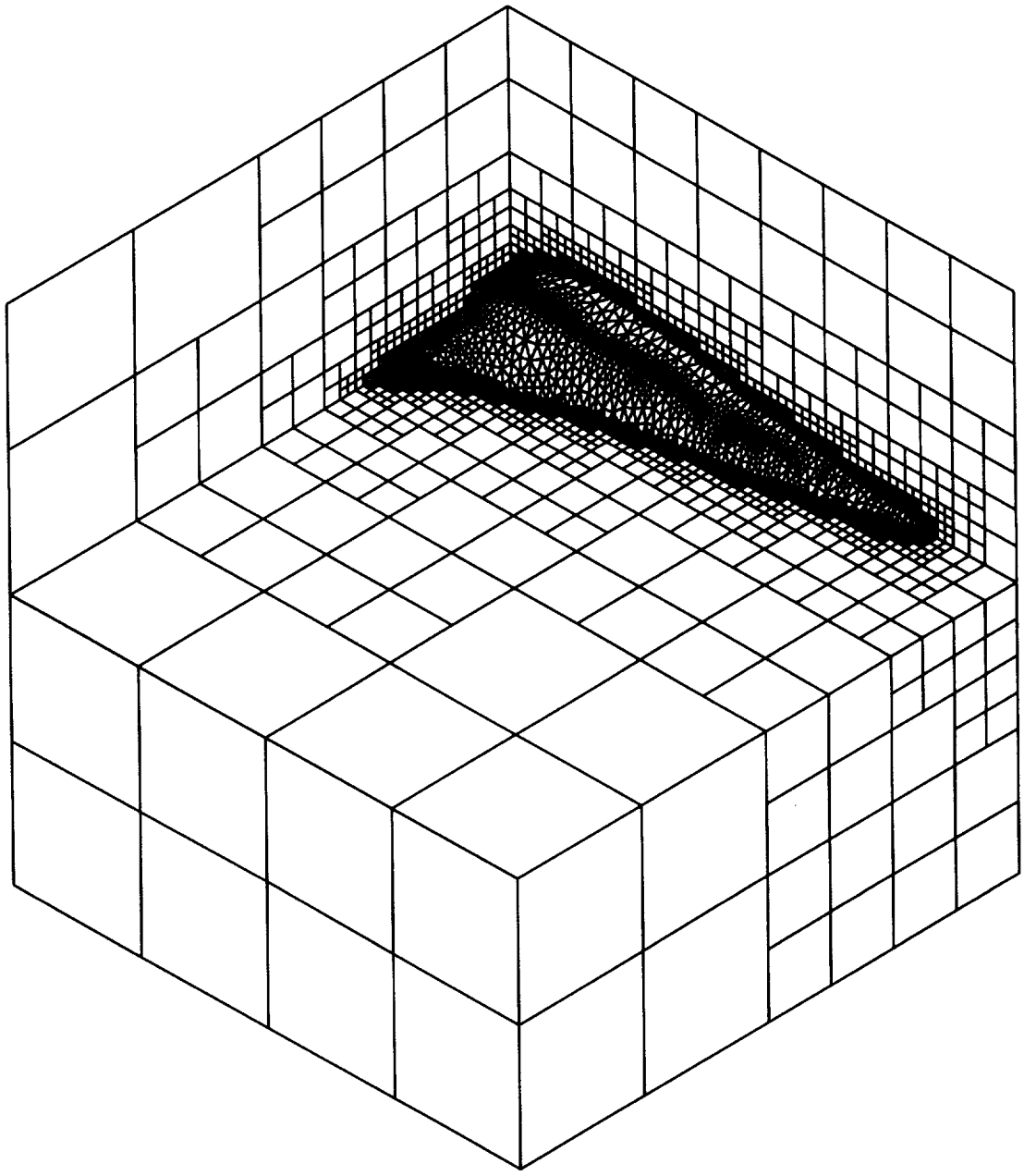


Figure 3: Outward refinement results in gradual growth of the octree.
3-D section of domain showing the outer prismatic surface and the octants.

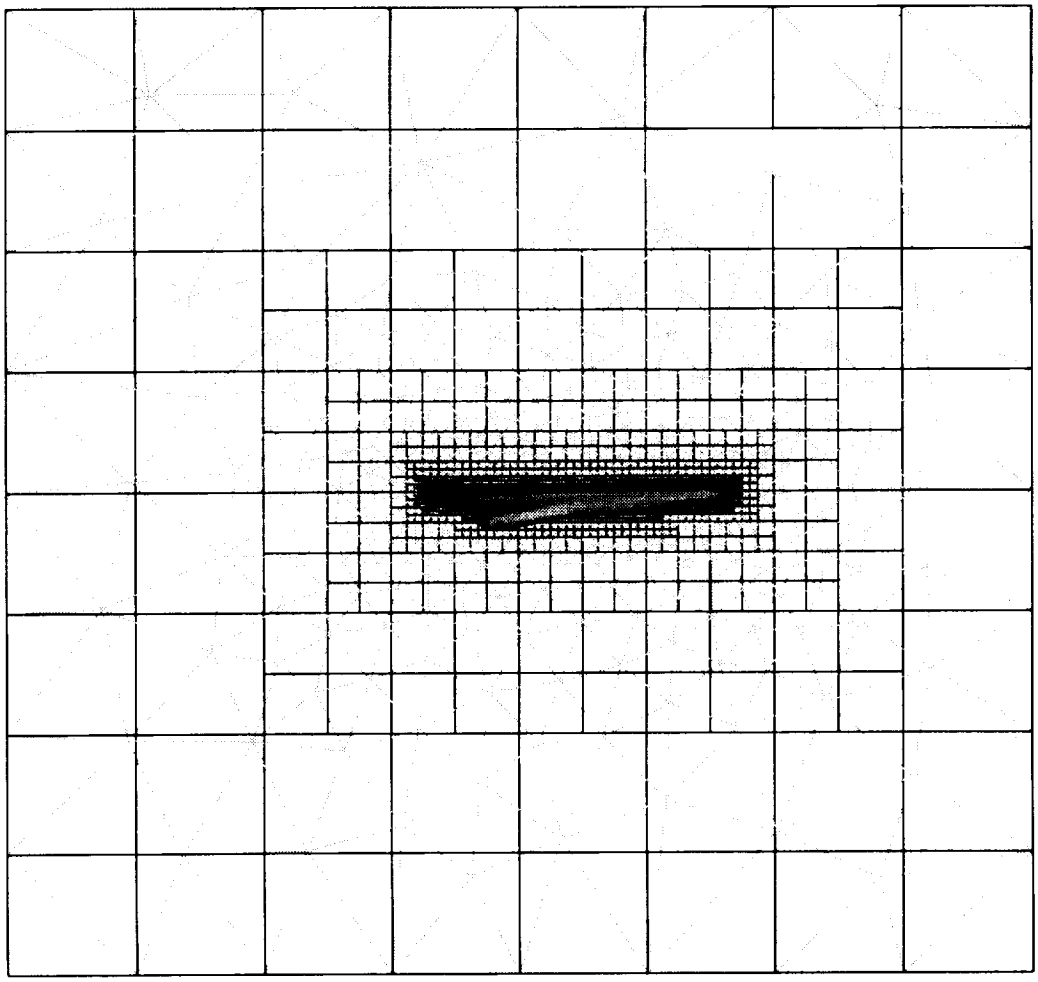


Figure 4: Effect of the octree on growth of the tetrahedra.
View of the octants (quadrilateral faces), as well as of the tetrahedra (triangular faces) on the symmetry plane. Growth of the tetrahedra away from the outermost prisms surface follows growth of the octree quite faithfully.

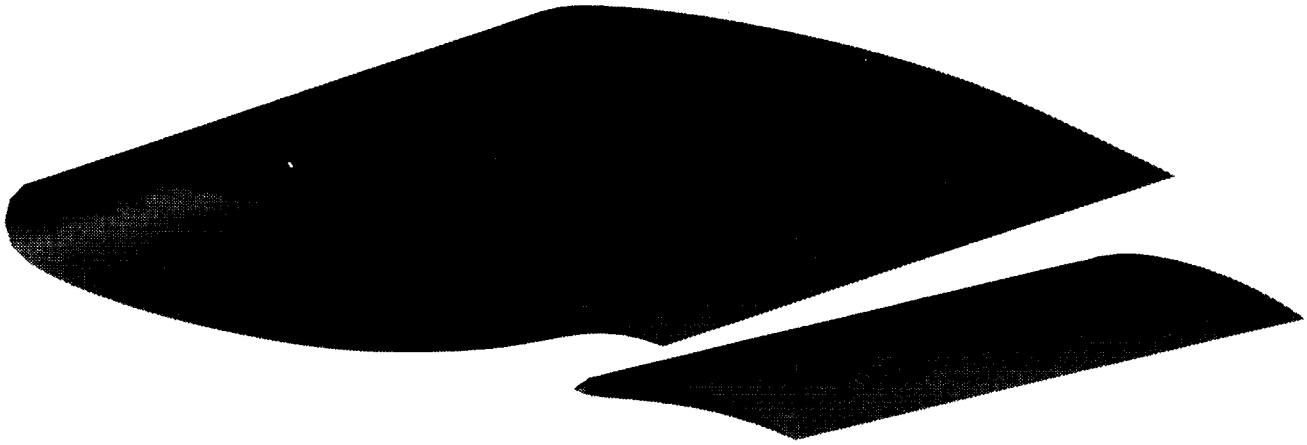


Figure 5: Geometry of two-element wing with a variable size gap between the main wing and the flap.

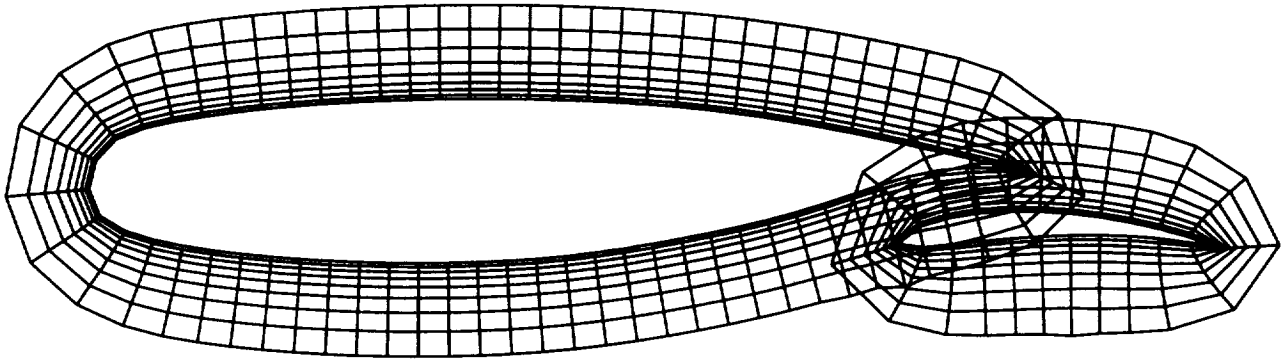


Figure 6: Prismatic grids grow around each body independently of one another (view on the symmetry plane).

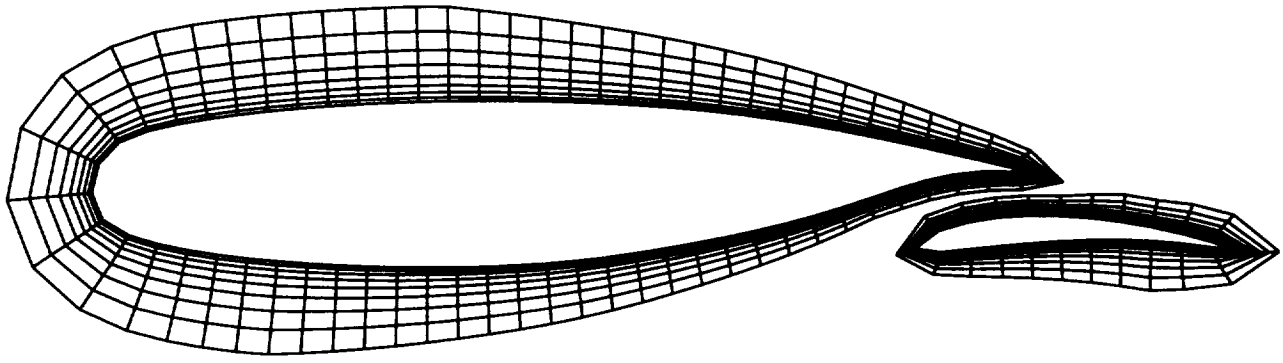


Figure 7: Mutual receding of the two prismatic grids removes prior overlapping (view on the symmetry plane).

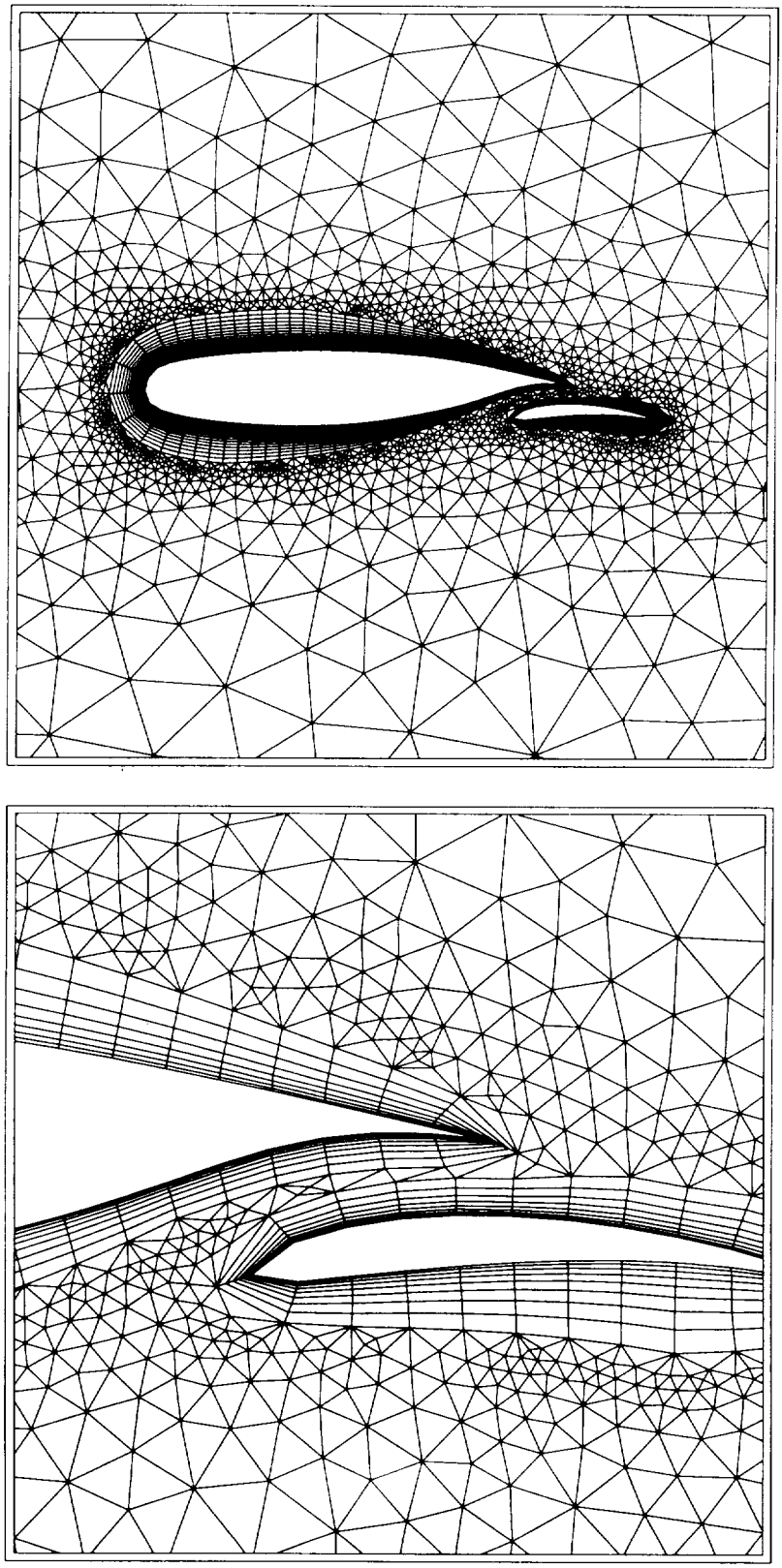


Figure 8: Tetrahedral grid fills the areas in between the two prismatic meshes (view on the symmetry plane).
Lower portion: Enlarged view of the gap region between the two bodies.

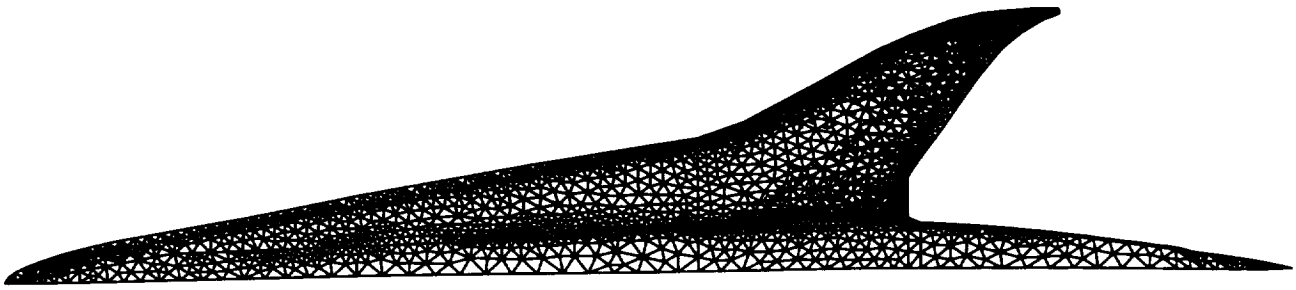


Figure 9: Triangulation of the HSCT surface (4412 triangles, 2275 nodes). A symmetry plane is considered.

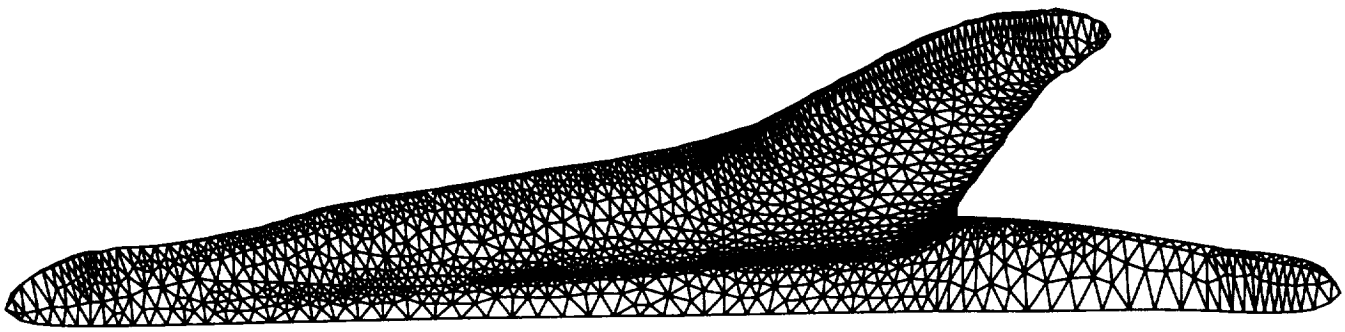


Figure 10: View of the outer surface of the prismatic mesh of the HSCT.

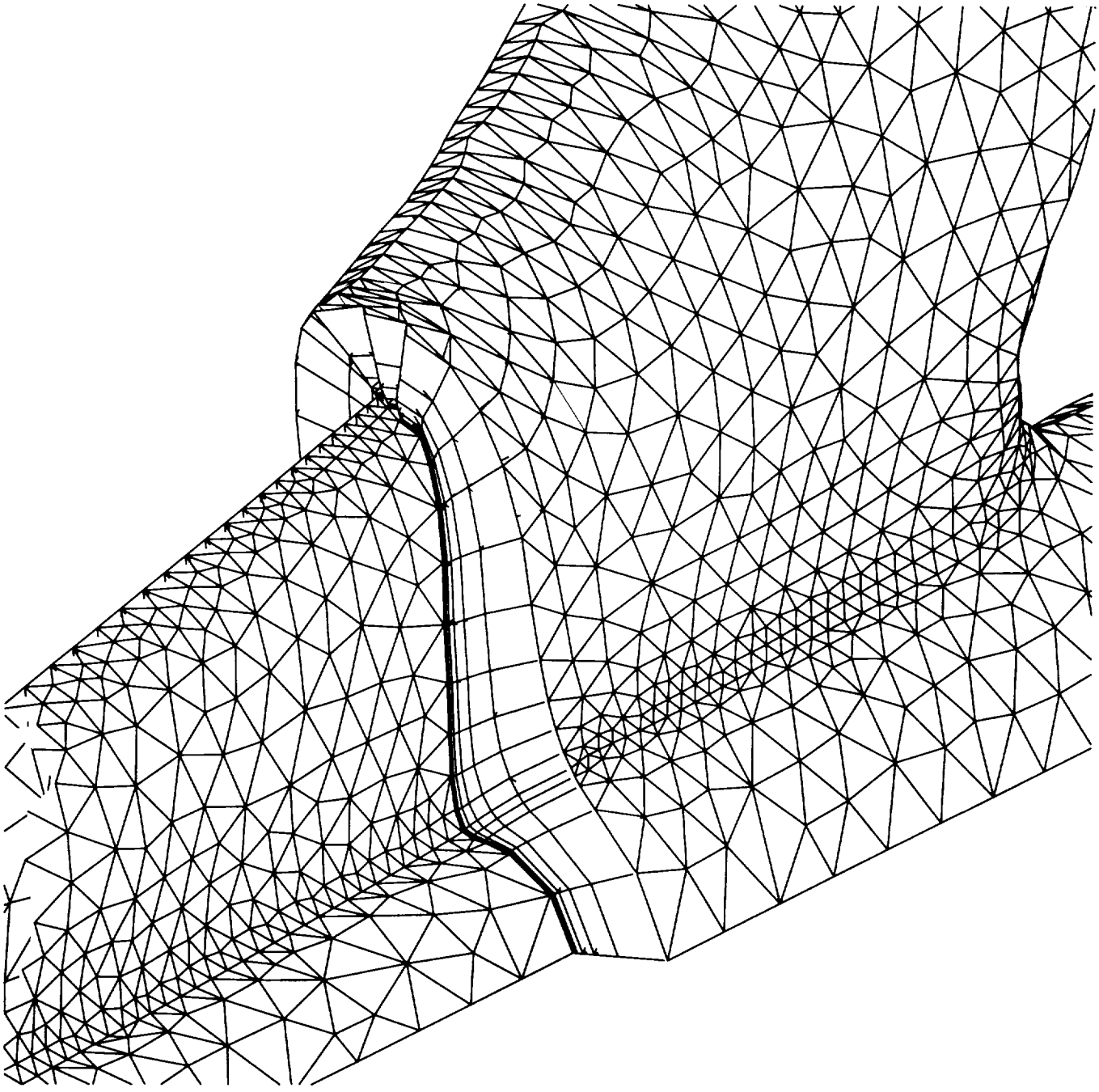


Figure 11: View of the initial and final prisms surfaces around the HSCT as well as of the quadrilateral faces of the prisms (every fourth layer is plotted).

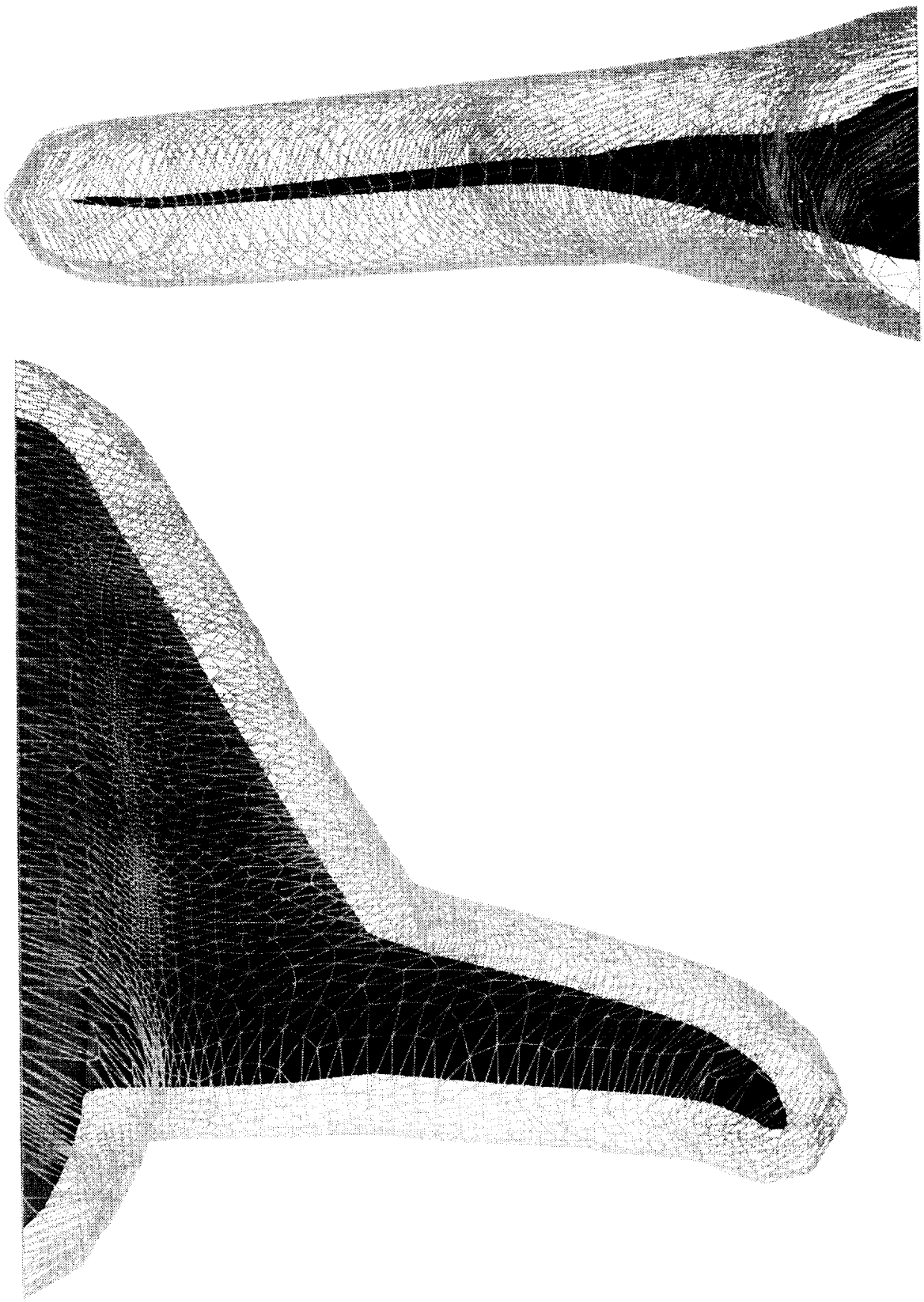


Figure 12: Wire-frame views of the grown prismatic mesh around the wing of the HSCT (shaded area denotes the surface of the aircraft).

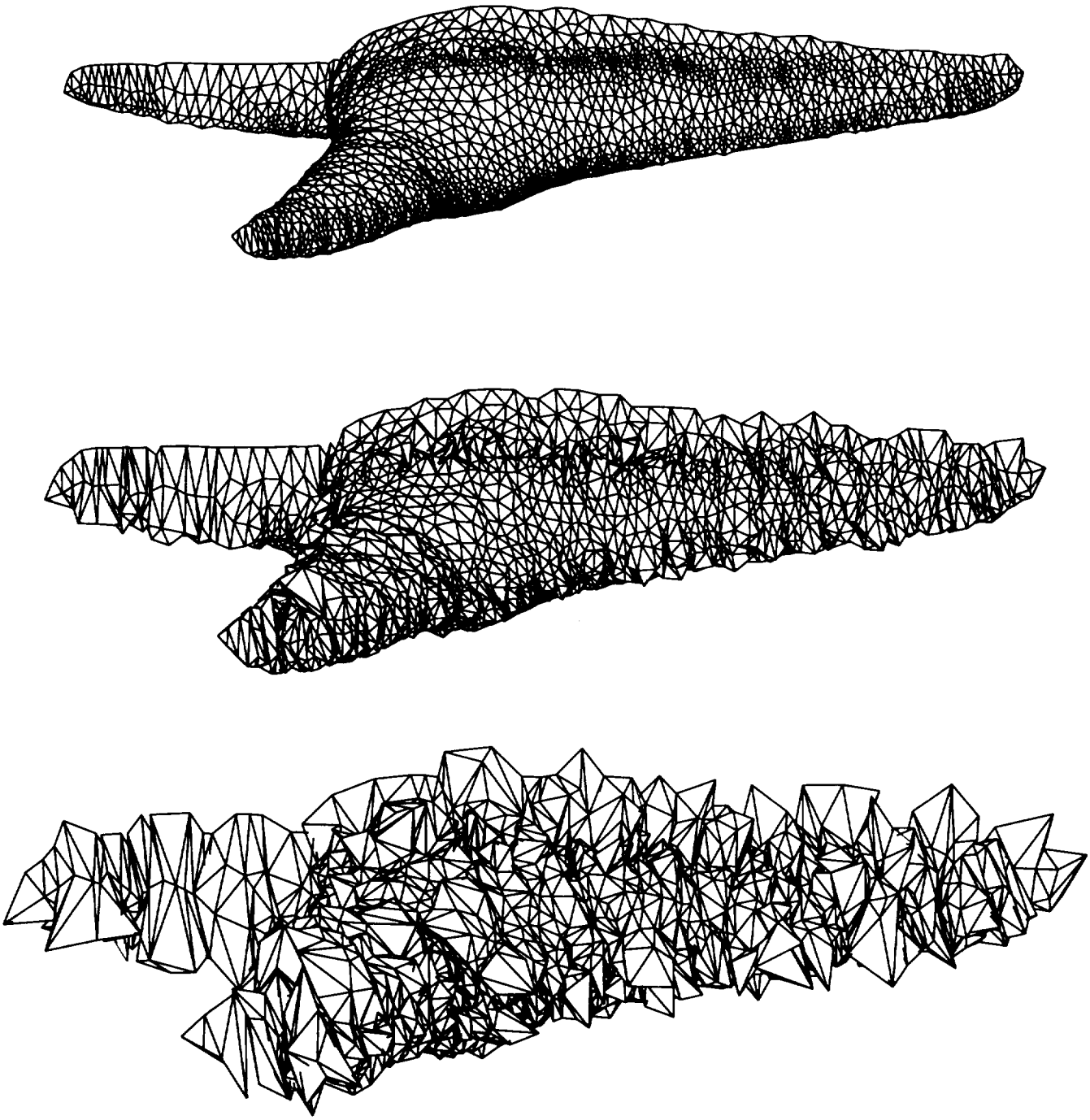


Figure 13: View of different stages in growth of the layer of tetrahedra away from the prisms outer surface for the HSCT.

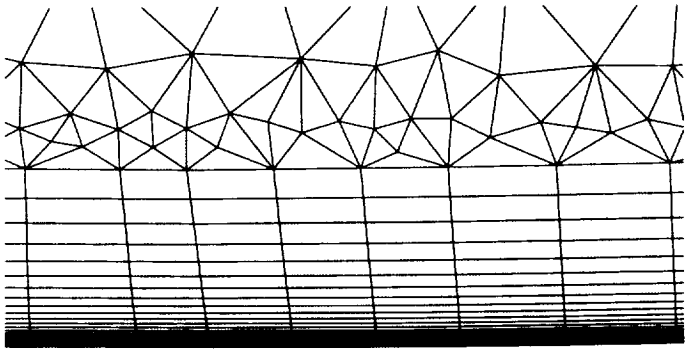


Figure 14: Smooth transition of the hybrid grid from the prisms to the tetrahedra. Partial view of the mesh on the symmetry plane.

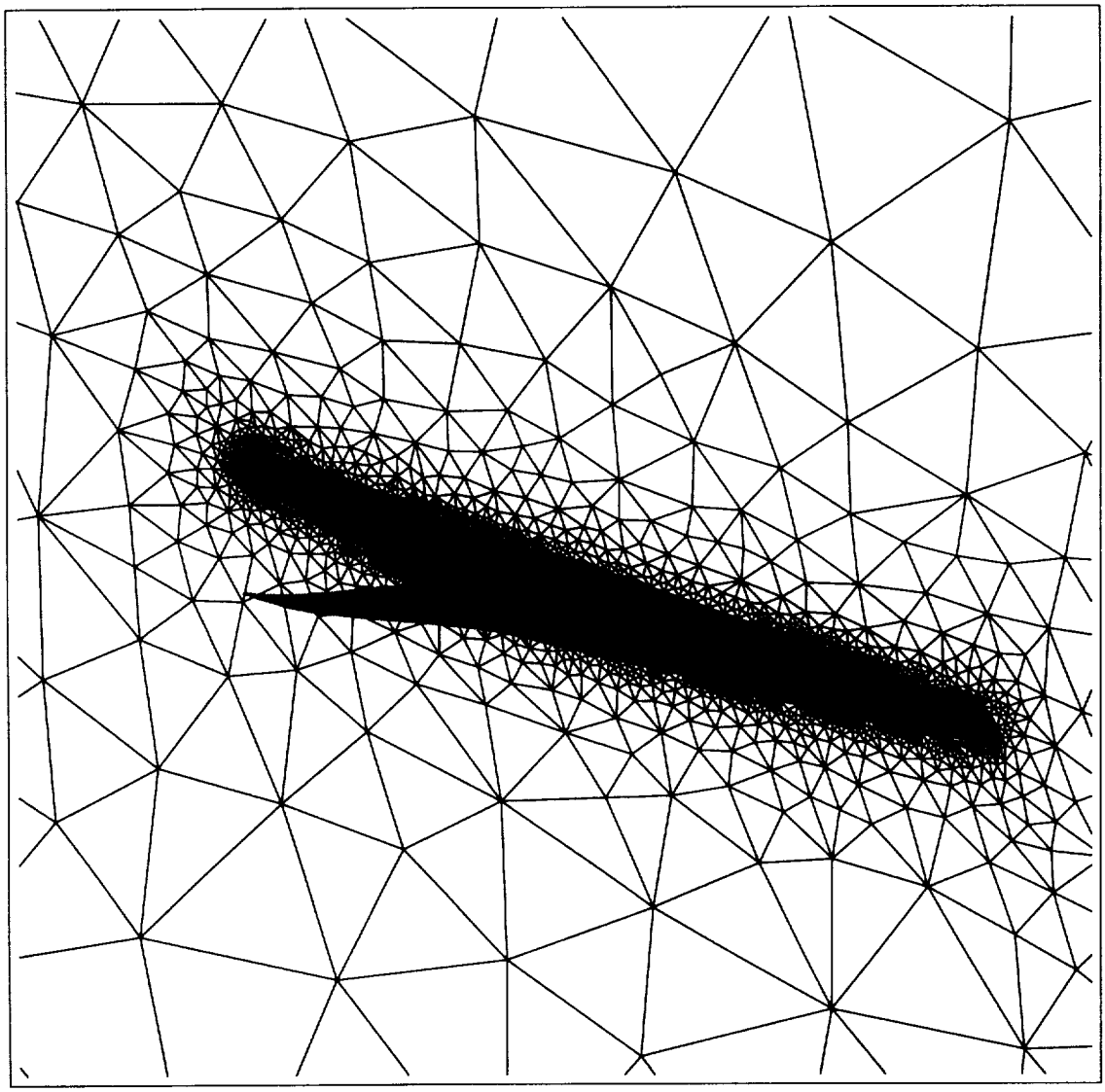


Figure 15: View of the hybrid grid on the symmetry plane of the HSCT. The quadrilateral faces correspond to the prisms, while the triangles belong to the tetrahedra.

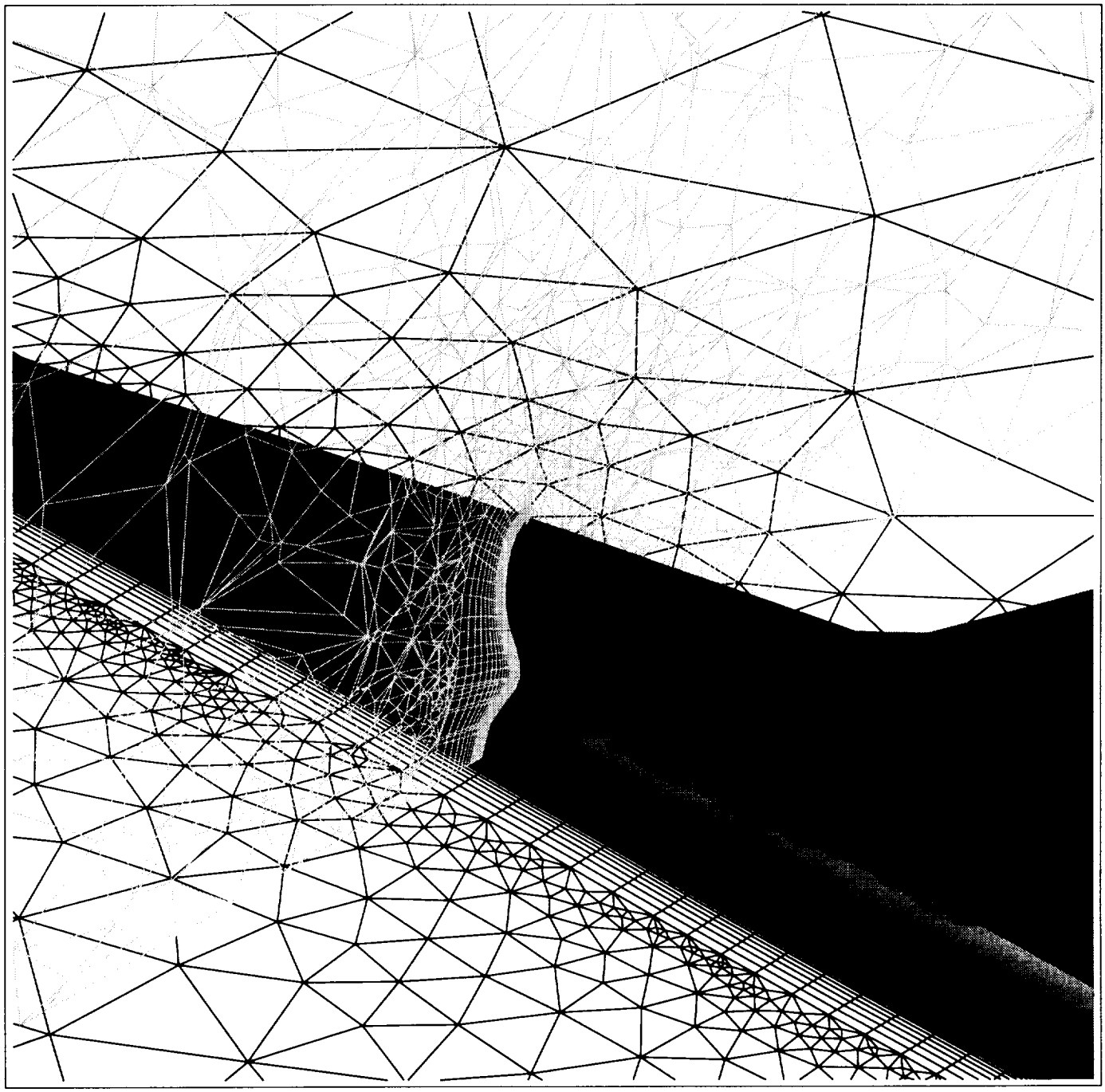


Figure 16: View of the hybrid mesh around the HSCT on two different planes that are perpendicular to each other. The first plane is the symmetry (dark lines), while the second is intersecting the fuselage at a location upstream of the wing (light lines).

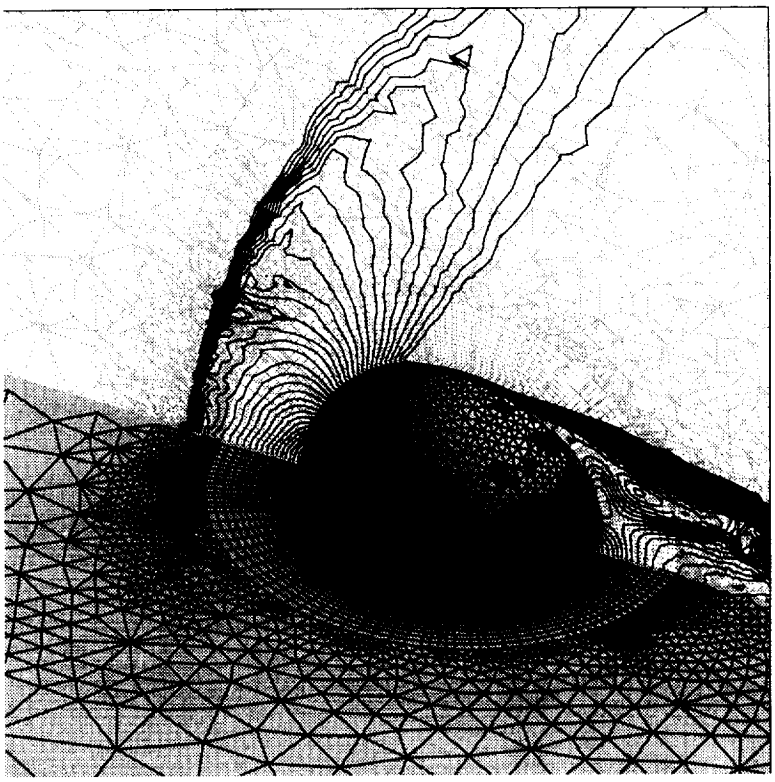


Figure 17: An isometric view of the tessellation on the wall surface, symmetry plane and an interior equatorial plane and Mach number contour lines on the equatorial plane. Hybrid grid embedded isotropically in the tetrahedral region and directionally in the prismatic region. ($M_{min} = 0., M_{max} = 2., \Delta M = 0.05$).

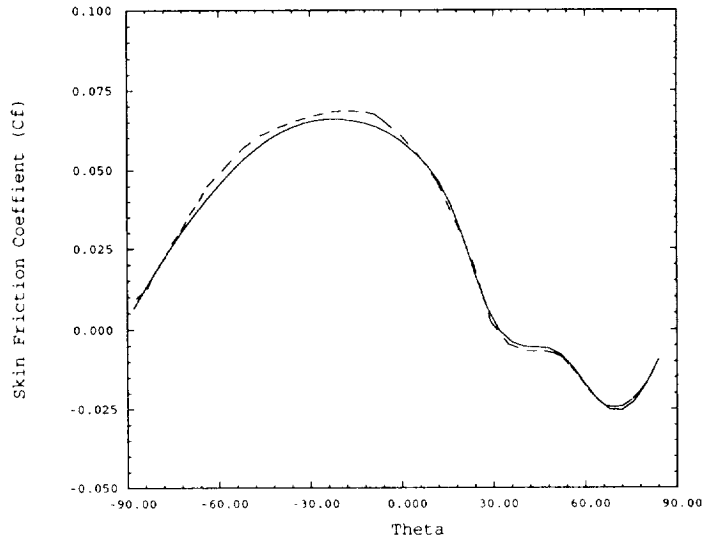



Figure 18: Comparison of Skin-friction coefficients at the wall, on the equatorial plane normal to the symmetry plane.
—— Globally refined hybrid grid solution, – – – Locally embedded hybrid grid solution.

N95-28745

# THREE-DIMENSIONAL HYBRID GRID GENERATION USING ADVANCING FRONT TECHNIQUES

John P. Steinbrenner and Ralph W. Noack
MDA Engineering, Inc.
Arlington, TX

## ABSTRACT

A new 3-dimensional hybrid grid generation technique has been developed, based on ideas of advancing fronts for both structured and unstructured grids. In this approach, structured grids are first generated independently around individual components of the geometry. Fronts are initialized on these structured grids, and advanced outward so that new cells are extracted directly from the structured grids. Employing typical advancing front techniques, cells are rejected if they intersect the existing front or fail other criteria. When no more viable structured cells exist, further cells are advanced in an unstructured manner to close off the overall domain, resulting in a grid of "hybrid" form. There are two primary advantages to the hybrid formulation. First, generating blocks with limited regard to topology eliminates the bottleneck encountered when a multiple block system is used to fully encapsulate a domain. Individual blocks may be generated free of external constraints, which will significantly reduce the generation time. Secondly, grid points near the body (presumably with high aspect ratio) will still maintain a structured (non triangular or tetrahedral) character, thereby maximizing grid quality and solution accuracy near the surface.

## INTRODUCTION

Grid generation has turned out to be a significant aspect of the computational simulation of field problems. A cursory literature search would produce a variety of fundamental grid types in mainstream use, which is an indicator that no one method offers clearly superior features. The traditional method of grid generation for computational fluid dynamics, structured grid generation, remains popular today because it provides substantial solution accuracy, particularly in viscous regions of the flowfield. A generalization of the structured method, the multiple block method, allows virtually any geometry to be modeled with a series of abutting grid blocks. This geometrical flexibility comes at a price, however, as the person-hours needed for generation can become exhorbitantly high. This one fact was probably the major reason for the ground swell of unstructured grid generation CFD applications in the last eight or so years. Unstructured grids do indeed ameliorate the generation-time bottleneck, but there has been a conspicuous delay in application of unstructured grids to viscous regions, where the severe length-scale disparities of the problem necesitates cells of very large aspect ratio. Even now that viscous applications are appearing in the literature, the memory and computational overhead associated with an unstructured grid makes it difficult to generate grids of sufficient resolution. In 3D, for example, it takes 5 or 6 tetrahedral cells near the body to replace a single hexahedral cell, and even then the tetrahedra will usually exhibit high aspect ratios.

Much of the grid generation research of the past few years has been aimed at alternate methods that allow grids to be generated in reasonable times without compromising resolution and hence, solution accuracy. Perhaps the most mature of these methods is the overset, or Chimera approach. Here, structured grids are generated independently about different components of the geometry in an overlapping manner. Overlap regions are then determined either from user-input, or lately from automated means [1], and solutions are obtained on the composite grid. The difficulty of this method is in the conservative interpolation of the flow solution from one component grid to another. Complex methodologies have been developed for this step. Another emerging grid technology is the cartesian approach, whereby the flow domain is automatically

divided into quadrilaterals and parallelepipeds at locations aligned with the physical coordinates. Here, local refinement is often used for enhanced accuracy. The current problem with the cartesian method is the representation of the geometry surfaces, which are usually not aligned with the physical axes. The typical remedy is to modify the cell locally near the surface to maintain a body-conforming fit.

This last fix introduces yet another emerging grid technology, that of the grid of mixed structure, or the "hybrid" grid. The most popular hybrid grid application is to combine regions of structured grids with regions of unstructured grids in such a way to preserve the salient features of each method. For example, by using structured grids near the geometry surface, and unstructured grids everywhere else, the inner grid could be generated quickly, and the outer grid could be generated automatically, thus eliminating the difficulty of constructing a multiple block topology that links several blocks together to represent the entire domain. The structured character of cells near the body would then lead to improved solution accuracy, while representing the domain locally with fewer grid cells. Early hybrid grid examples were applied to geometries with clearly defined near and farfield regions [2], with the non-overlapping structured grids generated first, and the remaining regions filled in later with unstructured cells. More recent examples, applied to extremely complex geometries, still require user-defined boundaries between regions of structured and unstructured cells [3]. In the general case, however, the line of demarcation between structured and unstructured regions is not clearly defined, and in fact, may be impossible for the user to specify. The most intuitive solution is to allow the near-geometry region to be generated automatically, until a specified distance is reached or until intersections are found. Very impressive grids have been developed recently using this approach [4], [5], but as the generation of surface and near-surface grids is the toughest aspect in *all* types of grid generation, it may be some time before grids of this type may be generated automatically for general geometries and flow conditions.

Another means of generating hybrid grids is to generate a series of structured, overlapping grids around different components of the geometry, similar to the chimera approach. Here, however, grid cells in the overlapping regions of the grid will be removed, and will be replaced with unstructured cells. The resulting hybrid grid will then fully encapsulate the flow domain with no regions of overlap. Determination of regions of overlap becomes the central issue in this approach. One effort employed well-established chimera tools for blanking out the regions of overlap between grids [6]. The chimera tools used, however, required the grids to be sorted into major and minor components, which forced users to be familiar with the manner in which the component grids overlapped. For obvious reasons, requisite user-comprehension of block overlap will impede the generation of hybrid grids around general configurations, and so an automatic means of overlap detection is desirable.

The method proposed in this paper for hybrid grid generation manages grid overlap automatically through the use of an advancing front algorithm. Specifically, fronts are formed on surfaces of the structured blocks, and are then advanced by covering structured cells that do not intersect the front.[†] The end result is a collection of structured cells that cover the flow domain in all regions except for the gaps where opposing fronts nearly converge on each other. The gaps are in turn discretized with unstructured cells by applying another advancing front solution, this time using unstructured cells.

The remainder of this paper explains the details of the advancing front hybrid approach, and provides both 2 and 3 dimensional applications.

[†]The basic idea of using advancing front techniques on a series of structured grids is also presented in Reference [7]. In that chimera grid application, the advancing front determined regions of the grid to be blanked out in the composite grid, and limited the overlap (a chimera requirement) rather than eliminated it.

## BASIC ADVANCING FRONT METHOD

Generation of hybrid grids in this study is based on advancing front methods traditionally used for unstructured grid generation. Hybrid generation consists of two successive applications of a general advancing front algorithm- the first advancing along a predefined set of overlapping structured grid data, resulting in a non-overlapping set of structured cells, and the second advancing along previously unfilled regions of the domain using unstructured methods. Despite the inherent differences in these two applications, each employs the same basic steps:

1. Form an initial front as a closed-loop of faces.
2. Identify a face (*pface*) on the front from which a cell is to be constructed.
3. Form a candidate cell, *cell1*, using the *pface*.
4. Test the candidate cell for validity.
5. If the candidate cell is valid, add the cell to the grid, and modify the front so that the new cell lies behind it. This requires removing faces that are covered by *cell1*, and adding the remaining faces of *cell1* to the front.
6. If the candidate *cell1* is *not* valid, find another candidate cell and go to step 3.
7. Repeat the process until there are no more faces on the front, or until there are no more suitable *pface*'s.

The specifics of the advancing front method for both structured and unstructured grid regions will be detailed in subsequent sections.

## COMMON DATA STRUCTURES AND ALGORITHMS

There are so many fundamental similarities between the two advancing front applications that the current method attempts to utilize common data structures and routines in generating the structured and unstructured grid regions.

### Data Structure

There are three basic geometric entities, each with an associated data structure. The smallest discretization of the overall problem domain is known as a *cell*, which represents an area in 2D and a volume in 3D. Cells are bounded by a number of *faces*, which represents a length in 2D and an area in 3D. The edges of the faces are then represented by either a single *node* in 2D, or a line bound by two nodes in 3D. Nodes are the basic geometric entity, and their data structure contains an $(x, y, z)$ coordinate triple indicating location in physical space. A face data structure, on the other hand, contains a list of forming nodes (2 in 2D, 3 in 3D) as well as a pointer to the two cells formed on each side of the face. Finally, a cell data structure is composed of a list of faces and nodes. The number of faces per cell depends on the dimension of the problem and on whether the cell is structured or unstructured. Triangular (2D) and tetrahedral (3D) cells are used for unstructured cells, and quadrilateral (2D) and hexahedral(3D) cells are used for structured cells. Figure 1 lists the number of faces required to form the different cells used in the present work. Note that there are twelve faces in the hexahedron cell because each hexahedral side is broken into two triangular faces in order to utilize the same face intersection routines (described below) as the tetrahedral mesh generator.

The final data structure to discuss is the *front*. The overall front is a list of *front* entities, each of which contains a face and the associated connection data that specifies the face's neighbors at each edge. Additional geometric data is stored to facilitate the various searches required by the method. A bounding sphere radius and center is stored for each cell and face on the front to assist in face intersection and proximity tests

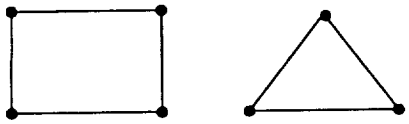
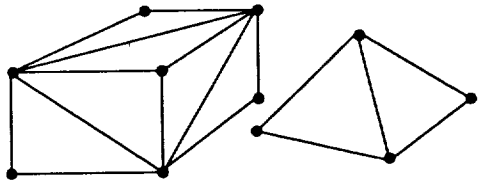
| 2D | | 3D | |
|---|---|---|---|
|  | |  | |
| quadrilateral | triangle | hexahedron | tetrahedron |
| 2 nodes/face | 2 nodes/face | 3 nodes/face | 3 nodes/face |
| 4 faces/cell | 3 faces/cell | 12 faces/cell | 4 faces/cell |
| faces are lines | faces are lines | faces are triangles | faces are triangles |

Figure 1: Cell structures for 2D and 3D.

described later. A front face also stores the normal to the face, area of the face, target cell size evaluated at the face centroid, and the angle between the face and its neighbors. A *state* indicator is also stored at each face on the front and reflects how close the face and one or more of its neighbors are to forming a complete cell. State values will be described in depth later in the structured and unstructured advancing front details.

## Quadtree/Octree Data Storage

The advancing front method requires a significant number of global data searches. Two examples include searching for a list of all nodes on the front that might be suitable for forming a cell, and searching for the list of candidate faces that might intersect a candidate cell. Typically several searches of this type are needed for every iteration of the advancing front. Clearly then, global linear searches for all of these possibilities is prohibitively expensive. The present approach to speed these searches is to use octree (3D) and quadtree (2D) data structures [8], so that linear searches are localized to a relatively small vicinity. Currently the nodes on the front and a list of faces containing the node are stored in one octree/quadtree structure, and the front faces themselves are stored in another, with the face centroid used as the storage location. The bounding box for the quad/hex containing the list of faces is expanded to contain the bounding sphere for all the faces contained in the quad/hex. Thus when searching for the list of faces whose bounding spheres intersect the bounding sphere of a cell, the cell bounding sphere is tested against the bounding box of a quad/hex in the tree. Those bounding boxes that do not overlap do not need to be tested further.

## Intersection Tests

The validity of a candidate cell, *cell1*, formed by advancing the front depends upon accurate testing for intersections between *cell1* and the faces on the front. The first step is to determine a set of faces on the front that are likely to intersect the faces of *cell1*. The bounding spheres for the candidate cell and the faces on the front are utilized in the search for a set of faces that that could possibly intersect the candidate cell. Since the cell and a face are completely contained within the bounding sphere, if the spheres do not intersect, then neither will the cell and faces. Hence rigorous intersection tests need only be applied to the list of all faces whose bounding spheres intersect the bounding sphere of the candidate cell.

The intersection test between a cell and a face begins by checking if the edges of the cell intersect the face and conversely, checking if the edges of the face intersect the faces of the cell. Additional tests are performed to reject the candidate *cell1* if the edges of the cell lie within the face or if the edges of a face lie within the

faces of the cell. Finally, *cell1* is rejected if any nodes on the front lie within the volume of the cell.

## HYBRID FLOW SOLVER

An extensive development effort parallel to this one is currently underway for a 2 and 3 dimensional Euler and Navier-Stokes solver for use with the hybrid grids generated herein. The flow solver adopts a cell-centered finite-volume formulation and an unstructured face-based grid connectivity that allows each cell to have an arbitrary number of bounding faces and allows each face to have an arbitrary number of forming nodes on its perimeter. The system of equations is solved with an implicit point Gauss-Seidel relaxation scheme. An upwind difference flux-vector splitting scheme is used to define the numerical flux at each cell interface. A second-order accurate upwind-biased extrapolation of the primitive flow variables defines the left and right states at the cell face using a cell-averaged gradient computed with a Green-Gauss integration of the solution reconstructed at the forming nodes. This flow solver is described in detail in [12].

## GENERATION OF STRUCTURED MULTIPLE BLOCK GRIDS

A major advantage hybrid grids afford the user over multiple block structured grids is the ability to generate block grids independently for different components of the geometry, thereby eliminating the unwieldy topological constraint of point-to-point matching at block boundaries. In this hybrid approach, these component block grids are assembled together, forming regions of single grid representation, regions of grid overlap, and regions of no grid representation. While these last two types would preclude the multiple block approach, both are handled with this hybrid algorithm, which automatically removes overlapping regions, and which fills gaps between grids with unstructured cells.

Still, the structured grid remains the starting point for the hybrid process, and serves as the "skeleton" for the final grid. In this method, structured grids are generated in multiple block clusters, with each cluster containing a series of blocks that abut one another exactly over a portion of the blocks' surfaces. The limiting case of single block clusters is permissible, so single blocks may be considered a subset of multiple blocks for this scheme. Fortunately, a multiple block grid cluster is sometimes as easy to generate as an individual block, and the quality of grid found in a multiple block cluster is worth preserving in the hybrid grid. For example, in Figure 2, the two abutting blocks ($D$ and $E$) around the main airfoil require no more work to construct (perhaps less) than would several overlapping blocks around the same airfoil. The proper strategy, then, is to generate multiple block grids only in regions where the topology does not markedly impact generation time.

Numerous tools are readily available for the construction of multiple block grids. All grids in this work were generated with Gridgen [9], [10], a widely-used general-purpose software package designed for use with a suite of public-domain flow solvers. Gridgen automatically detects regions of block-to-block connections, and also provides a graphic user-interface for the establishment of flow solver boundary conditions on block surfaces. Further, it exports ASCII grid data and connection data that is directly readable by the selected flow solver. This feature was exploited in the development of the hybrid solver, which adopted Gridgen's generic file formats for both grid point and connection data. This allows the user to set up the majority of the hybrid grid generator inputs directly in Gridgen. Keep in mind, however, that any multiple block grid generator would work as well, once the exported data were translated to the proper format.

## HYBRID GRID GENERATION

The generation of the hybrid grids from the predefined set of structured grid points follows a linear path that begins with converting the structured grids into hybrid components, and continues with parsing these points into non-intersecting regions, forming an initial front, advancing the front along structured cells,

removing unused cells, massaging the front and then advancing it further using unstructured cells.

## Importing Grids into Hybrid Structure

The hybrid algorithm commences with a conversion of the raw grid data generated in multiple block form into the node-face-cell paradigm required by the hybrid solver. Conversion to the hybrid format follows the multi-step path described below.

For the trivial single block case, the remainder of this section is unnecessarily complex, since connections will not exist, and the block's nodes, faces and cells could be generated automatically on the basis of their implied connectivity. In the more general (and more interesting) multiple block case, however, surfaces from different blocks will abut one another, with any number of block edges and corners sharing nodal values. In these cases it will not suffice to treat blocks independently, lest multiply defined nodes and faces be formed on the block's abutting regions.

Construction of the hybrid entities begins with node formation, and continues with face and then cell formation for *all* blocks in the system. No consideration is given to component overlap or intersections during this phase. Rather, the entire structured grid is fit into a hybrid structure that will later serve as the template on which the advancing front algorithm will be applied. This later phase will determine overlap and will act accordingly.

Connection Arrays.- The interblock connection file associated with the multiple block grid is read into a series of arrays as a preliminary step. These arrays specify flow boundary conditions or connections with other blocks for each bounding surface of the block. Connections and BC's may be applied at the partial surface level, meaning that more than one BC or connection may be applied to a particular block boundary. Next, blocks in the system are divided into groups based on topological linkages defined in the connection arrays. Two blocks of the same group indicate that it is possible to get from one block to the other while crossing only at connections specified in the arrays. Figure 2 illustrates the block group concept for a 6 block, 3 group example. Although block groups make no guarantee about the relative geometric positioning of the blocks, as a matter of practice it is assumed that blocks of the same group contain no region of overlap. This is not unreasonable in consideration of the fact that block groups will generally correspond to components of the geometry, and will likely be generated independently of other groups. This assumption of no overlap within block groups will be exploited later to reduce the number of intersection tests during front advancement.

Formation of nodes.- Nodes are the most difficult hybrid component to construct from a set of multiple-block structured grid data because there exists no one-to-one correspondence between grid points and nodes. In Figure 3, for example, grid points from three individual blocks are all represented by the same hybrid node value. In contrast, at most two structured grid surfaces will be represented by a hybrid face, and a hybrid cell will always represent a unique region of a structured grid.

To facilitate the placement of nodes into face and cell components, a temporary rectangular array of Null-initialized node pointers $node_ptr[i,j,k,n]$ is formed for each block in the system, sized to the computational dimensions of the block. The $i,j,k,n$ indices correspond directly to the structured grid point with like-indices, with $n$ referring to block number. The $node_ptr$ array is filled easily for indices on the interior of the block, since interior points will correspond to unique nodes. For these indices, a new node is allocated, and its address is assigned to $node_ptr[i,j,k,n]$. For indices corresponding to block extremities, however, more care is taken. If the value in $node_ptr$ is Null, a list of *all* grid points using the node is formed, a new node is allocated, and its address is assigned to $node_ptr$ at each grid point index in the grid point list. On the

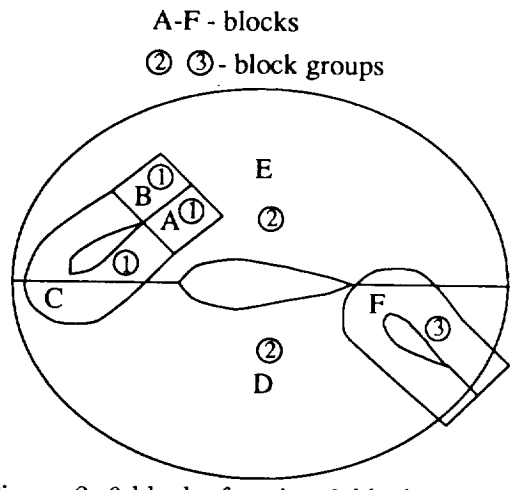


Figure 2: 6 blocks forming 3 block groups.

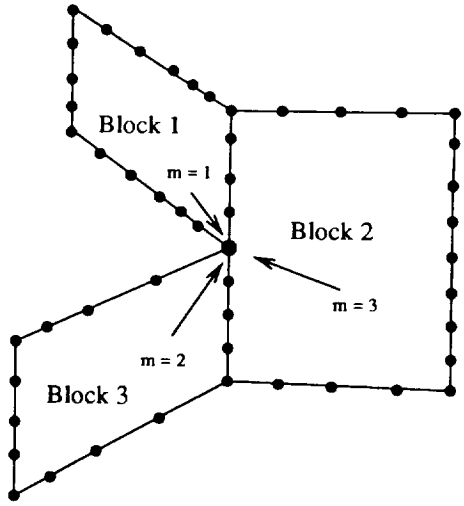


Figure 3: Nodes common to blocks.

other hand, if the value in *node_ptr* is not Null, a node is already assigned to the grid point, and the next index is considered.

The list of block grid points sharing a node is determined via an iterative algorithm that traverses the outer surfaces of the structured blocks, searching for minimum surface elements that use the node. A minimum surface element is defined here as a 2 by 2 patch (2 by 1 in 2D) of adjacent grid points on the block boundary. Define also a *POF* (point on face) as an artificial entity containing $i, j, k, n$ ranges of a minimum surface element, and indices to one of the element's corners.

Now, for each grid point $i, j, k, n$ index on the outer surfaces of the structured blocks, a *POF*, labeled $A$, is formed at the index. $A$ is pushed onto a stack, and the iterative loop begins.

1. Look through the stack for the first unused *POF*, called $P$. Mark $P$ as used. If there are no unused *POF*'s, exit.
2. Using the connection arrays, determine if $P$ abuts to another block. Call this "image" *POF* $I$. If it exists, check $I$ for uniqueness with all other *POF*'s on the stack. If unique, add it to the stack, and mark it used.
3. There will be up to 3 other *POF*'s on the block boundary that will be adjacent to $P$ and will share the same grid point index (up to 1 *POF* in 2D). Add each one of these "neighbor" *POF*'s to the stack if they are unique, and mark them as unused.
4. Similarly, look for up to 3 other *POF*'s on the block boundary that are adjacent to $I$ and share the same grid point index. Add each one of these "neighbor" *POF*'s to the stack if they are unique, and mark then as unused.

When this algorithm is exited, *all POF*'s corresponding to a particular hybrid node will be determined. The final step is to scan the list of *POF*'s one more time, removing any *POF*'s that correspond to the same $i, j, k, n$ grid point index. What is left is a list of all block grid points that wil share the same nodal value. This algorithm is illustrated in Figure 4, which tracks the 4 iteration history of determining the grid points associated with a particular node.

| | | | | | |
|---|---|---|---|---|---|
| Get the first unused POF on the stack, called P.<br>Mark as USED. | A | D | E | F | - |
| Using connection info, find the image of P, called I.<br>Mark as USED, add to stack if unique. | B | - | C | - | |
| Find neighbor(s) of P sharing node i on the boundary.<br>Mark as UNUSED, add to stack if unique. | D | A | B | C | |
| Find neighbor(s) of I using node i.<br>Mark as UNUSED, add to stack if unique. | E | - | F | - | |

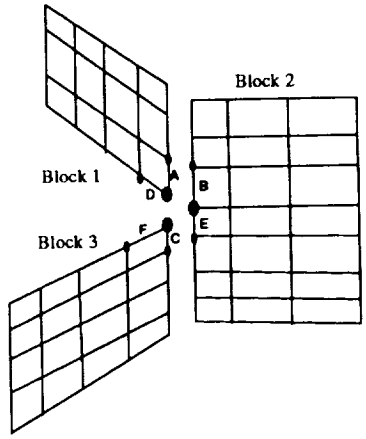



Figure 4: Algorithm for determination of common nodes.

Formation of cells.- Hybrid cell formation from structured grid data proceeds by allocating a temporary rectangular array of Null-initialized cell pointers $cell_ptr[i,j,k,n]$ for each block in the system, sized to the cell dimensions (grid point dimension minus one) of the block. Each array is then filled by allocating a new cell and assigning its address to $cell_ptr$. Nodes are then assigned to these cells by using the $node_ptr$ data at the same $i,j,k,n$ index in addition to its neighbors $((((node_ptr[i+a,j+b,k+c,n], a=0,1), b=0,1), c=0,1))$

Formation of faces.- Face formation also proceeds by traversing the $cell_ptr$ arrays at all possible cell indices $i,j,k,n$. The cell data structure contains pointers to all 12 (4 in 2D) of its constituent faces. At a given $i,j,k,n$ index, heretofore unassigned faces are allocated and assigned to the cell. Faces interior to blocks are also assigned to the cell adjacent in the $cell_ptr$ arrays. Faces on the boundary of blocks are checked for abutment with other blocks via the connection arrays. If such a connection exists, the newly allocated face is assigned to the abutting cell.

Treatment of singularities.- In order to fully represent a component of a geometry with structured multiple blocks, it is often convenient to introduce regions of singularities, or poles. Grid points lying along a 3D axis of symmetry, for example, will locally collapse to a region of zero area in the circumferential direction. Fortunately, these regions are easily identified and may be set with a Pole-type boundary condition in the connection arrays.

During node formation above, a node is allocated for each point along the singularity axis within the specified Pole BC regions and is assigned to $node_ptr$. This $node_ptr$ value is then applied to all remaining grid points in the direction of the singularity, thereby preventing the duplication of nodes. During face formation, then, if the candidate face accesses a given node more than once, the face will have zero area, and will be discarded and removed from the cell(s). This in turn modifies the local form of the cell, which could be reduced from 12 to as few as 4 faces in 3D, or from 4 to 3 faces in 2D.

## Identification of Non-Intersecting Subblocks

All structured grid points have now been loaded into the hybrid entities, namely the nodes, faces and cells. In the next section, some of these faces will be grouped and linked to form initial fronts for the advancing front algorithm. As will be shown, the front is advanced by swallowing one cell at a time, employing a series of intersection tests to ensure that the new front does not intersect the previous front. This is a relatively expensive test that is applied to the complete front once for each non-intersecting cell, and one that is clearly

not needed if it is known *a priori* that the cell in question does not violate intersection tests.

In light of this, a simple preprocessing step based on rectangular-extent or bounding-box testing [11], is employed to identify regions of the grids quickly that do not overlap or intersect with any others. Figure 5 depicts a collection of non-intersecting cells determined purely from these tests, described below.

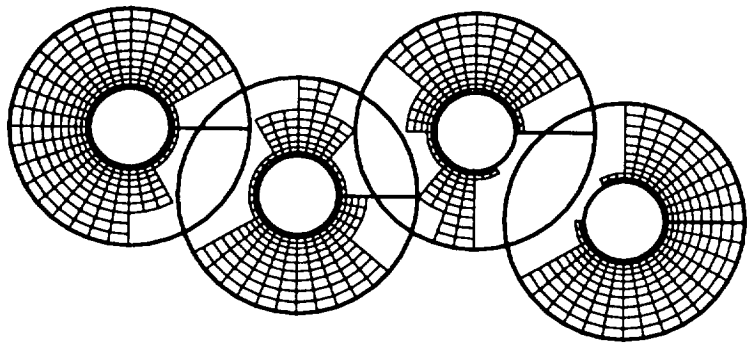


Figure 5: Non-intersecting subblocks.

Formation of subblocks.- The subblock entity is a data structure containing the computational and physical extents of part of a block. Initially, one subblock is formed for each block in the system, extending over the entire block, from 1 to $(imax, jmax, kmax)$. Each grid point in the subblock is then examined to formulate the range of $x$, $y$ and $z$ values for the subblock (the bounding box).

Splitting subblocks.- Next, a number of independent subblocks is constructed through successive subblock splitting. A subblock is deemed independent if its bounding box does not intersect the bounding box of any other subblock with a differing block group number. The test for subblock intersection is trivially performed due to the cartesian shape of the bounding boxes.

When a subblock is found to intersect another, it is split along its longest computational dimension, and new bounding boxes are computed for each subblock. The subblock is split as long as it does not violate a user-prescribed (usu. 3) minimum size. If an intersection is not detected, the subblock is independent and flagged as such. The procedure ends when there are no more dependent subblocks left to split.

Concatenating subblocks.- The previous step generally produces a large number of small independent subblocks, and so the subblocks are concatenated in this step to reduce their number. Concatenation takes place by checking the computational range of each edge of each subblock for an identical range elsewhere in the list. A match indicates two fully abutting subblocks, which are combined into a larger subblock. The procedure continues until there are no two remaining subblock edges that represent the same surface.

Remove overlapping subblocks.- Finally, all subblocks flagged as dependent are removed, resulting in a list of block ranges on which there is known to be *no* overlap or intersection. All cells within these independent subblocks are marked for easy disposition later.

## Formation of Initial Front

The collection of newly formed cells, faces and nodes represents a superset of the structured entities that will comprise the hybrid grid. This superset will be pared down to a non-intersecting set of entities by advancing a series of initial fronts one cell at a time, rejecting cells along the way that violate overlap and

proximity criteria. There are two types of initial fronts- those specified by the user and those representing bounding surfaces of non-intersecting subblocks formed earlier. Both types of fronts must form closed surfaces, so that it will remain simply connected as it covers cells. Such a requirement will eventually allow some of the front to disappear, signaling the end of the algorithm.

The faces in the system which will comprise the initial fronts are specified in a list of structured surface regions. User-specified front regions are passed in through the connection arrays, and generally will correspond either to the geometry surfaces or to the farfield extents of the overall domain. A total of six (4 in 2D) regions (corresponding to the block boundaries) are also added to the list for each remaining subblock. Subblock boundaries are chosen as initial fronts because they represent boundaries of regions of no overlap, and subsequent front advancement will move away from the boundaries, thereby eliminating a large number of unnecessary intersection tests.

Initial fronts are formed in two sweeps. In the first, each surface region, which represents a computationally rectangular surface on the structured blocks, is swept. Each face in the region will be loaded into the front, and will be connected to the faces with adjacent $i, j, k, n$ index, via the *cell_ptr* arrays. When the region is loaded, only the region's bounding edges will not be connected to other faces.

Special care is needed for pieces of front regions that are doubly-defined. This situation occurs at common surfaces of subblocks, and at subblock surfaces also user-specified as initial fronts. In all cases, a doubly-defined front face should be removed from the front, and so if a face to be added is already on the front, it is instead removed, and its neighbor's connections are broken.

In the second sweep, the edges of front regions and doubly-defined regions are stitched together to ensure a series of simply-connected initial fronts. This is accomplished by interrogating each edge of each front face. If the neighbor to the face at the edge in question is not defined, a search is made to find all faces on the front using the edge. If two edges are returned, their corresponding faces are linked together, and the next edge is found. The case when more than two edges are returned indicates an improperly formed initial front, at which time the algorithm is aborted.

Temporary arrays *cell_ptr* and *node_ptr* may be discarded after front initialization. From this point on, all connectivity between cells, faces and nodes is transmitted from within their respective data structures.

## Advancing Front Along Structured Cells

Except for the user-specified initial front selection. all of the steps described above proceed in an automatic fashion. When these steps have completed, the advancing front algorithm is started along the structured cells. Front advancement follows an iterative three step procedure until the front has disappeared or until it may be advanced no more. In the first step, a suitable face on the front is selected for front advancement. Secondly, the cell in front of this face (the cell to be covered) is tested for geometric violation with other faces on the front. Finally, if it passes the test, the front is modified to include the cell, and the front face connections are modified to maintain a proper front.

Candidate face selection.- The first step in the iterative structured front advancement procedure is to select a face, called *pface*, from which the front is to advance. The importance of *pface* selection should not be underestimated, as differing selection criteria may produce hybrid grids of widely varying character. This is because front advancement at a given location precludes advancement of an opposing front that is vying to cover the same physical space.

Penalty function: Since the proper selection of *pface* is influenced by a host of conflicting geometrical

and topological factors, it is logical to assemble these factors into a mathematical "penalty" function, so that *pface* may be selected as the face of minimal functional value. In this work the linear expression below is used, where $n$ refers to the number of component functions, $c_i$ is a user-set non-negative influence coefficient, and $f_i$ is the normalized $i'th$ component function.

$$F_{penalty} = \sum_{i=1}^{n} c_i f_i$$

There are currently two component functions used in the overall penalty function. The first, $f_{size}$, measures the relative size of the face. Specifically,

$$f_{size} = \frac{log(S/S_{min})}{log(S_{max}/S_{min})}$$

where $S$ is the area (length in 2D) of the face, and $S_{min}$ and $S_{max}$ are the minimum and maximum areas among all faces on the front. Notice that $f_{size}$ is bound by 0 and 1. This function produces a smaller penalty for faces of smaller size, but the logarithmic form reduces order of magnitude disparities in face size to a linear scale, preventing over-penalization of the mid-size faces.

The second component function, $f_{state}$, measures the local "raggedness" of the front. Here, a "state" variable is formed at each face in the cell, set equal to the number of front neighbors that touch the same cell outside the front. Front faces with larger values of the state variable result in a local "stair-step" construction of the front, as depicted in Figure 6. This local phenomenon is to be avoided, if possible, because it reduces the advantage of local structure in the hybrid grid, and it creates a more challenging starting front for the unstructured grid generator described later. Therefore, the goal of $f_{state}$ is to minimize the penalty at faces that tend to fill in the ragged edges of the front. This suggests the following form:

$$f_{state} = 1 - \frac{T}{T_{max}},$$

where $T$ is the state value, and $T_{max}$ is the maximum state value found on the front.

In practice, $f_{size}$ seems to be the more important function, and is typically used with coefficients of $c_{size} = 1$ and $c_{state} = 0.25$ . This combination insures that the *pface* will be chosen on the basis of state variable when the face sizes are nearly the same.

By no means are the two functions above sufficient for front face selection in all cases; this is in fact one of the lesser mature aspects of the advancing front hybrid approach. Other penalty function components for future consideration include the proximity of the front face to opposing fronts and the variance of the forming cell size from a prescribed target volume.

Cycling: During the early stages of front advancement, it is desirable to have the front grow in a fairly uniform manner, so that viscous layers of the structured grid may be covered by the front before overlap in the grids becomes a factor. This is accomplished by associating a *cycle* variable to all front faces, roughly equivalent to the number of layers from the initial front that the current face lies. A global *cycle* value is also formed, equivalent to the layer that is currently being filled. As the front advances, only front faces with the current value of *cycle* become candidate *pface*'s, and the winning *pface* is selected from the candidates using the penalty function approach described above. After *pface* is covered by the front, any new front faces are given an incremented *cycle* value so that they will not become candidate *pface*'s until the next layer (global *cycle* is incremented). When there are no more front faces with the global *cycle* value, the *cycle* value is

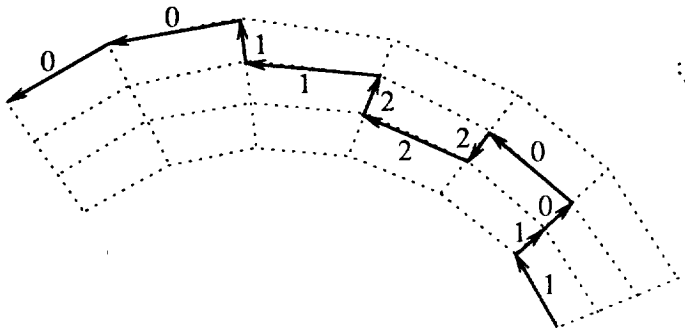



Figure 6: State values along a typical front.

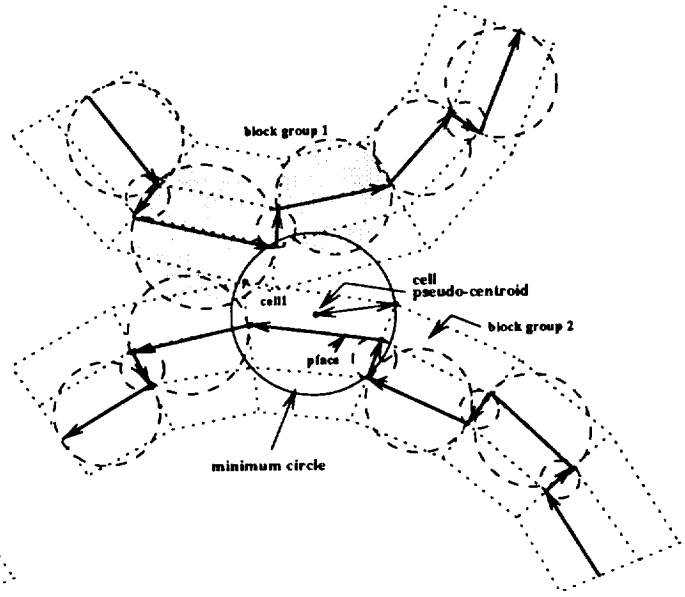



Figure 7: *cell1* is rejected due to proximity to shaded circles.

incremented, and the procedure is repeated. When the number of user-specified cycles is met, all front faces become candidate *pface*'s henceforth.

A typical value for *cycle* is 3, which guarantees 3 layers of grid points around initial fronts, barring front violations, described next.

Violation checks with front.- Before the structured front is advanced at a given *pface*, the faces of *cell1*, the cell outside the front at *pface*, are checked for two types of geometric violations, described below, with the rest of the front. If a violation is detected, the front is *not* advanced at *pface*, and *pface* is flagged so that it will not be a candidate face in subsequent iterations.

Strict intersection: All faces on *cell1* not already on the front are checked for intersections with all faces on the front as described earlier. If an intersection is found, *cell1* clearly *may not* be covered by the front.

Proximity: Experience has indicated that it is also necessary to keep fronts from getting "too close" to each other. This is driven by the fact that the completed structured front will serve as the starting front for the unstructured generator, and a finite distance between fronts is needed to provide adequate space for the unstructured front to form reasonable cells.

Definition of the term "too close" is understandably nebulous. The approach herein is to calculate the pseudo-centroid of the *cell1* by averaging the node values, and then to form a sphere (circle in 2D) of minimum radius that surrounds all nodes. Next, for each face on the front, the sphere (circle in 2D) of minimum radius that contains all nodes on the face is calculated. The front face minimum spheres are then scanned for intersections with the minimum *cell1* sphere, ignoring front faces belonging to the same block group as the original *pface*. Front faces of the same block group are ignored because front faces adjacent to *pface* would almost always be intersecting, and it is implicitly assumed that blocks of a common block group contain no overlap.

This type of "proximity" check is schematically illustrated in Figure 7, and its effect is shown in the

example of Figure 8.

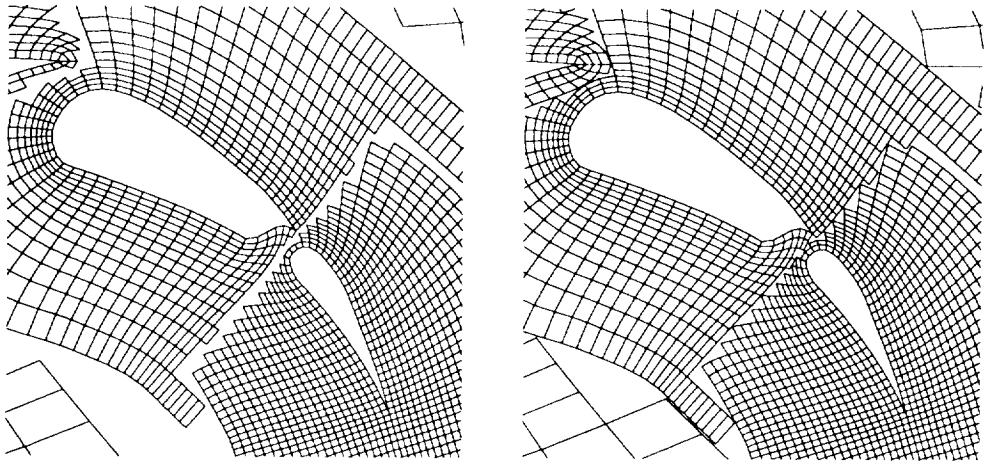


Figure 8: Structured cells advanced with (l) and without (r) proximity tests.

Front modification and reconnection.- If the *pface-cell1* combination passes both violation checks, the front is next modified to cover the cell. First, the cell to be covered, *cell1*, is flagged to indicate that it now lies behind the front. Next, faces in *cell1* not already on the front are added, and faces in *cell1* already on the front are removed. Finally, the new faces are connected to one another and to the neighbors of faces removed from the fronts. This task is straightforward for simple cases, but complicates quickly for odd combinations of converging fronts. For that reason, it was necessary to develop the general algorithm for front reconnection outlined in Figure 9.

| edge → | n1 | n2 | n3 | n4 |
|---|---|---|---|---|
| Get a list of all faces and Cell1's (cells outside of front) on front using the edge. | f1,c1<br>f2,c2 | f2,c2<br>f3,c3 | f6,c4<br>f7,c3 | f4,c1<br>f5,c4 |
| Remove faces in Cell from list. | f1,c1 | f3,c3 | f6,c4<br>f7,c3 | f4,c1<br>f5,c4 |
| Add non-front faces of Cell using the edge to the list. | f8,c1 | f10,c3 | f10,c3<br>f9,c4 | f8,c1<br>f9,c4 |
| Pair sets of faces with the same Cell1. Remaining pair (if any) is also paired.<br>For each pair, find corresponding edges. Link Face_a,Edge_a to Face_b,Edge_b. | f8-f1 | f3-f10 | f6-f9<br>f7-f10 | f4-f8<br>f5-f9 |

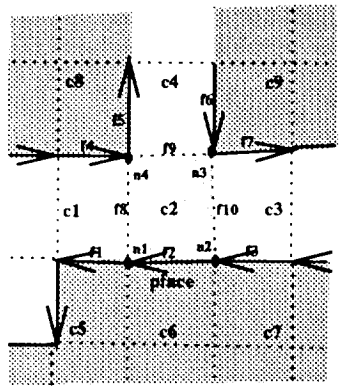



Figure 9: Front reconnection algorithm.

Front faces are reconnected via a four step procedure applied to each edge of *cell1* (*c2* in Figure 9), the cell outside of *pface* to be covered by the front. In 2D (Figure 9), this amounts to a reconnection at the four nodes of the cell. In the first step a list of all faces and their corresponding *cell1*'s is formed from all front faces that use the edge. Next, any face/cell combinations with cells equal to the original *cell1* (*c2*) are

removed from the list. Third, any of the faces on *cell1* that do not lie on the front are added to the list. There will now be an even number of list entries, and these entries are grouped together on the basis of like cell numbers. On occasion when there remains two unmatched entries in the list with different cells, they are matched to one another. These paired cell/face combinations now refer to the connections needed to maintain a proper front linkage. For example, in the first column (edge $n1$) in Figure 9, faces $f8$ and $f1$ must be set to neighbor each other along their common edge, $n1$.

In the figure above, connections at edges $n3$ and $n4$ are broken in the upper left and upper right fronts and are reconnected with the faces in *cell1*. This is contrary to intuition, which would have the upper left and upper right front connections remain intact, and the faces on *cell1* connect to one another. Repeated front reconnections like this would result in fronts that collapse on themselves like a deflated balloon. In contrast, the algorithm outlined above will produce collapse-free fronts that will continually divide and reduce to minimal surface area. The difference is crucial when applying the unstructured advancing front algorithm described shortly.

## Unused Component Removal

The structured portion of the advancing front algorithm finishes when each face on the front either lies on the outer edges of the multiple block grid or may advance no further due to close proximity to other regions of the front.

All cells, faces and nodes lying outside of the front at this point may be discarded, since they will not comprise part of the hybrid grid. Unused cells are determined by the absence of the flag set in cells overtaken by the front. After unused cells are removed, all faces and nodes in the remaining cells are flagged, and then all unflagged faces and nodes are removed.

## Diagonal Swapping on the Final Structured Front

Before the regions encapsulated by the final structured front(s) will be filled with unstructured cells, it is necessary to massage the front to reduce the burden on the unstructured solver. In 3D, a side of a structured cell lying on the front will always be represented by a pair of complementary triangular faces, originally formed purely on the basis of their computational index. Since the unstructured solver has a natural bias to triangles of smaller maximum angle, the diagonal along each face pair on the front is compared to the imaginary diagonal along the two non-common nodes on the two faces. If the latter diagonal is shorter than the former, the quad region is retriangulated with diagonals swapped, so that the maximum angle on both triangles is reduced, as shown in Figure 10.

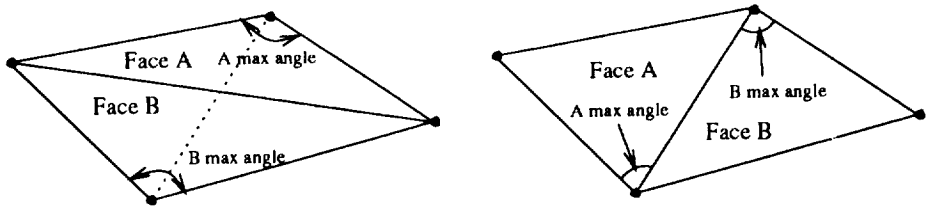



Figure 10: Swapping diagonals of complementary faces.

It is also helpful to swap diagonals on regions of the final front that represent inside corners of cells outside the front. Specifically, whenever three (or more) adjacent non-complementary faces on the front have the same *cell1* and share a common node, an inside corner situation exists, as depicted in Figure 11a. In these cases, the diagonals of the faces are swapped as necessary so that they do not touch the inside

corner. This creates a natural crevice that will later be filled with a tetrahedron by the unstructured solver (Figure 11b), thereby smoothing a formerly sharp corner of the front.

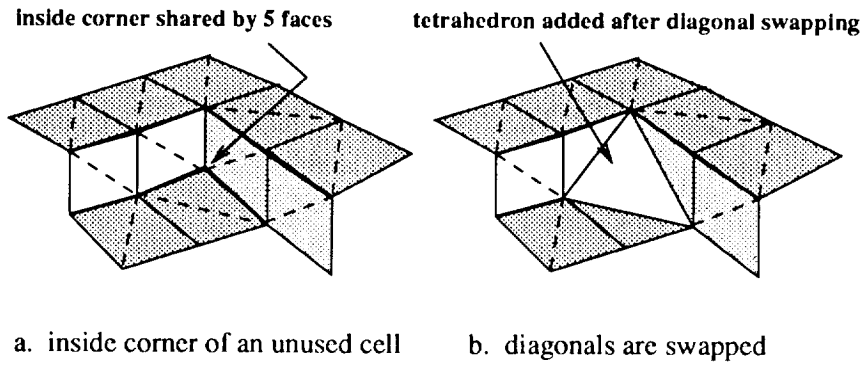



Figure 11: Swapping diagonals on inside corners.

## Refinement/Derefinement of Final Structured Front

The use of overlapping stretched structured grids to produce hybrid grids can produce starting fronts for the unstructured grid generator with significant variations in face sizes. When adjacent faces of a highly stretched structured grid cell are present on the front, a small face and a much larger face will be present. In addition a fine mesh around one component in the grid system may extend into the coarse mesh region of another component. Disparate sizes of faces in close proximity can lead to the generation of undesirable highly stretched or skewed triangles and tetrahedra. To lessen the face size disparity, local refinement and derefinement of the final structured front is added as a tool for further preprocessing the front for unstructured front advancement.

The derefinement procedure will merge faces on the front and their corresponding cells whenever the sizes of faces on the front are too small in comparison with a target size and whenever the aspect ratio of the structured grid cell behind the front is larger than a user specified value. Once the derefinement of the front is complete, the derefinement is propagated into the interior of the mesh my allowing a side of a cell to have at most two neighboring cells.

The refinement procedure will split faces on the front if the sizes of faces on the front are too large in comparison with a target size or if the size of a neighboring face. The procedure to split the cells behind the front has not yet been implemented.

## Advancing Front With Unstructured Cells

When the the front has advanced as far as it can along structured cells, and after the final front has been massaged as described in the previous two sections, discretization of the domain is completed by advancing the front again, this time until it completely vanishes.

After the first advancing front application, there remains no more usable structured grid data beyond the front, and so in the second application the front is advanced from the more traditional, unstructured sense. Unfortunately, there are a whole new set of concerns to address during the unstructured application, since the front will now advance on the basis of geometric, rather than topological reasoning.

State variable.- Recall that a state indicator is stored at each face on the front, and its value reflects how

close the face and one or more of its neighboring faces are to forming a complete cell. For unstructured grid advancement, a minimal state value means that none of the neighboring faces are suitable for forming a cell with the face. The next state level means that a single neighboring face is suitable for forming a cell. This occurs when the angle between the two faces is less than a user-specified threshold, typically 135 degrees. The next highest state level occurs when two faces touching the face share a node and the angle between the build face and the two neighbors is less than the threshold. The highest state level occurs when the face and all its neighbors form a complete cell. Different front face states for a triangle and a quadrilateral are shown in Figure 12. States for the 3D cell forms, the tetrahedron and hexahedron, are defined in an analogous fashion.

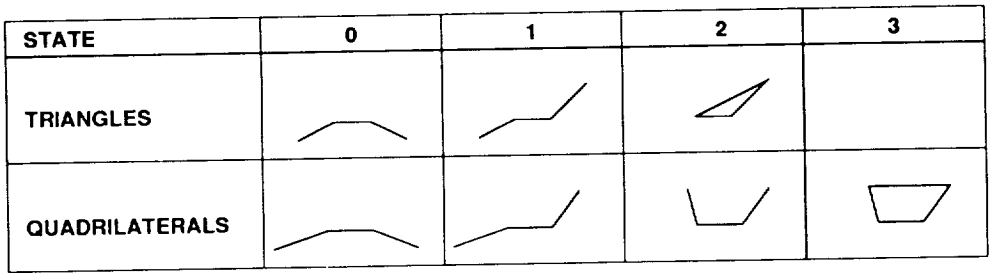



Figure 12: State levels for unstructured cells.

Target Face Size.- The present study uses a set of clustering points to specify a desired, or target, size for a cell face. Each clustering point is given by a spatial location, the desired size of a face, $d_i$ at that location $(x_i, y_i, z_i)$ and a clustering parameter, $C_i$, that controls the spatial variation away from the point. The target size, $t$, at an arbitrary point in space $(x, y, z)$ is given by

$$t = \min_i d_i + C_i R_i^2$$

where $R_i^2 = (x_i - x)^2 + (y_i - y)^2 + (z_i - z)^2$ and $\min_i$ represents the minimum value for all the clustering points.

The clustering points can be specified by the user or set to the size and locations of the faces on a set of boundary surfaces. The latter method ensures a smooth variation in faces away from each boundary with a minimum of user inputs.

The target size can also be specified by interpolating field values from a cartesian background grid similar to the approach of [13]. This would be the preferred approach for 3D cases due to the large number of faces on the surfaces of the geometry.

Candidate face selection.- Just as it was for structured front advancement, the unstructured grid generated by the advancing front method is significantly influenced by the criteria used for *pface* selection. During the unstructured application, the advancing face *pface* is set equal to the first face returned from the ordered set of criteria below:

1. the first face that is at the maximum state (forming a closed cell with neighboring front faces) or the maximum state minus one (lacking only one face to form a closed cell).
2. the face that forms the minimum angle (below the threshold angle) with a neighboring front face.
3. the smallest face on the front.

A degree of uniformity and smoothness may be incorporated into an unstructured mesh by assigning a cycle number to each front face and by advancing a face only if it has a lower cycle number than currently being used. As described earlier, when no suitable faces are found, the cycle number is incremented. This cycling strategy seems to work best when filling large areas/volumes that have smooth boundaries. Hence, it is probably not appropriate for use in this application- namely, filling in the jagged regions between overlapping structured grids.

Candidate Cell selection.- The most critical step in unstructured front advancement is the determination of the validity of the candidate cell, *cell1*. Candidate cells will still be deemed invalid if their faces intersect any others on the front, but they will also be rejected if they lie too close to others on the front. Rejection is necessary to filter out highly stretched or skewed cells from the hybrid grid.

The four step procedure below is currently used to arrive at a suitable cell emanating from *pface*, the building face.

1. If *pface* is at the maximum state, form the cell from its neighboring faces.
2. Generate a node perpendicular to *pface* using the target cell size for the face. If the node is not too close to another face, if the node is not within the bounding sphere of a face on the front, and if the faces formed by the node do not intersect the front, form the cell from *pface* and the generated node. In 2d, try to form an even better cell using Delaunay criteria as suggested by Merriam [14]. Use the node that is closest to the circumsphere center of the candidate cell to form a new candidate cell.
3. Use the nodes on neighboring faces that raised *pface* to its state to form a cell.
4. Form the cell by using nearby nodes on the front within some radius of *pface*.

Note that no attempt is made to maintain a Delaunay [15] grid. Though the Delaunay criteria will produce a grid with triangles that are the most equilateral, it is overly restrictive in 3D as it produces flat or planar cells in regions of coplanar nodes. Since the structured grids forming the initial unstructured front will typically have numerous regions of planar nodes, the Delaunay method would connect these nodes to form flat cells.

The nearly planar nodes and faces forming the initial unstructured front are especially troublesome when the faces surround a point that was a corner of a structured cell. See for example Figure 13. If the wrong choice for a candidate cell is made and accepted, the cell may well contain a face that is nearly planar with other faces on the front. A different choice for the starting face and hence the candidate cell would have eliminated the possibility of the cell with the planar face from even being considered. The present method begins by finding all the nodes with 6 faces surrounding the node. If two neighboring faces in the list for a node are found to be planar, then a cell is constructed from one of the faces. The same procedure is then applied to nodes with 5 and 4 surrounding faces.

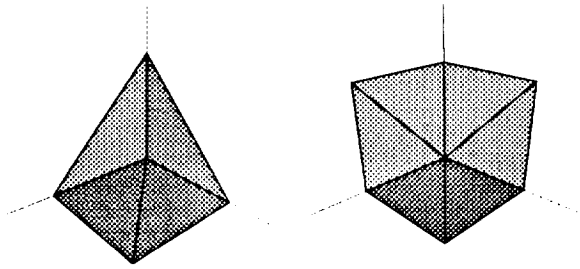


Figure 13: Node Surrounded by 4 faces(l) and 6 faces(r)

## EXAMPLES

The present approach is first illustrated using a two-dimensional multi-element airfoil as shown in Figure 14. Figure 15 presents the overlapping structured system, which consists of an "O-type" structured mesh around the main airfoil section and "C-Type" meshes around the leading and trailing edge flaps. The structured mesh with the overlapping cells removed is shown in Figure 16. Close up views of the leading and trailing edge regions are shown in Figure 17 and clearly illustrate the difficulties in using the front left by removing cells from a stretched and clustered mesh. The cells in the leading edge region of the main airfoil grid are stretched and have a high aspect ratio. The front will have a small face adjacent to a larger face, corresponding to the sides of one of the stretched cells. In the trailing edge region the fine mesh around the last trailing edge flap has cells, and hence faces on the front, that are in the coarse region of the main airfoil grid. Clearly it is advantageous to fill the void with a smoothly varying unstructured mesh.

Two different approaches have been tested in the present effort and are illustrated in Figures 18 and 19. The first approach uses refinement of the unstructured grid starting front to ensure a smooth variation in the faces on the front. Figure 18 shows that some of the sides of the quadrilaterals adjacent to the unstructured cells have been split into smaller faces without splitting the cell. The companion flow solver was developed to allow an arbitrary number of faces per cell. Refining the front may add too many cells so a second approach of derefinement has been tested. Figure 19 show the results for the case where the structured grid front and the cells behind the front have been derefined to provide a smoother distribution of cell sizes. Figures 20 and 21 show the final complete hybrid grids for the multi-element airfoil when refinement and derefinement strategies are used. The derefinement procedure has significantly reduced the number of cells in comparison to the refinement case. The preferred approach will obviously depend upon the particular flow conditions in the region of refinement or derefinement.

The results for a three-dimensional case are illustrated for a aircraft store in close proximity to a pylon. For simplicity the wing geometry is not considered in this example. The pylon and store geometries are shown in Figure 22 and a portion of the overlapping structured grids around the geometries is shown in Figure 23. The portion of the front resulting from removal of the overlapping portion of the pylon mesh is shown in Figure 24. and for the removal of the overlapping portion of the store mesh is shown in Figure 25. In this particular case the unstructured grid starting front will include a portion of the store surface since the store and pylon are in such close proximity. The tetrahedral unstructured mesh filling the void between the store and pylon meshes is not presented here due to the difficulty in viewing three-dimensional unstructured grids.

## CONCLUSIONS

This paper has introduced a new 3-dimensional hybrid grid generation technique based on advancing front ideas applied to both structured and unstructured grids. The advantages of the scheme have been shown to be that structured grids may be generated around individual grid components independently, with any overlap or gaps between grids removed and/or filled automatically. This dramatically reduces grid generation time, but still provides a locally structured grid near the geometry surfaces. Further, the structured character of cells near the body markedly reduces the number of cells needed to resolve the flowfield.

Future effort is aimed at improving the efficiency of the hybrid generation time and also the quality of the grid. Efficiency issues include the improvement of the quadtree/octree searching routines, and grid quality issues include modifications to the face selection criteria, refinement of the front proximity definition for converging fronts, and extension of the scheme to allow non-enclosed initial fronts, such as in problems with a plane of symmetry.

## ACKNOWLEDGEMENTS

This work was sponsored by the United States Air Force Wright Laboratory, Armament Directorate, Eglin AFB, FL under contract F08630-93-C-0074.

## REFERENCES

1. Belk, D.M., and Maple, R.C., "A New Approach to Domain Decomposition: The Beggar Code," Proceedings of the 2nd Overset Composite Grid and Solution Technology Symposium, October, 1994.

2. Weatherill, N.P., "On the Combination of Structured-Unstructured Meshes," **Numerical Grid Generation in Computational Fluid Mechanics '88**, Sengupta et al., eds., Pineridge Press, 1988.

3. Shaw, J.A., Georgala, J.M., May, N.E., and Pocock, M.F., "Appliction of Three-Dimensional Hybrid Structured/Unstructured Grids to Land, Sea and Air Vehicles," **Numerical Grid Generation in Computational Fluid Dynamics and Related Fields**, N.P. Weatherill, et al., ed., pp. 687-698, Pineridge Press, 1994.

4. Marcum, D.L., "Generation of Unstructured Grids for Viscous Flow Applications," AIAA Paper 95-0212, January, 1995.

5. Kallinderas, Y., Khawaja, A. and McMorris, H., "Hybrid Prismatic/Tetrahedral Grid Generation for Complex Geometries," AIAA Paper 95-0211, January, 1995.

6. Kao. K.-H., and Liou, M.-S., "Direct Replacement of Arbitrary Grid-Overlapping by Nonstructured Grid," AIAA Paper 95-0346, January, 1995.

7. Wey, T.C., "Development of a Mesh Interface Generator for Overlapped Structured Grids," AIAA Paper 94-1924, 1994.

8. Samet, H.: "The Quadtree and Related Hierarchical Data Structures", **Computing Surveys**, 16(2), pp. 187-260, 1984.

9. Chawner, J.R., and Steinbrenner, J.P., "Automatic Structured Grid Generation Using GRIDGEN (Some Restrictions Apply)", Proceedings of the 1995 Workshop on Surface Modeling, Grid Generation, and Related Issues in CFD Solutions, NASA CP, 1995.

10. Steinbrenner, J.P., and Chawner, J.R., **The GRIDGEN Version 8 Multiple Block Grid Generation Software**, MDA Engineering Report 92-01, 1992.

11. Foley, J., van Dam, A., Feiner, S., and Hughes, J., **Computer Graphics: Principles and Practice, Second Edition**, Addison-Wesley, 1990.

12. Bishop, D.G.; and Noack, R.W.: "An Implicit Flow Solver With Upwind Differencing For Three-Dimensional Hybrid Grids, " AIAA Paper 95-1707CP, 12th AIAA Computational Fluid Dynamics Conference, 1995.

13. Pirzadeh, S., "Structured Background Grids for Generation of Unstructured Grids by Advancing-Front Method," **AIAA J.**, 31(2), pp. 257-265, February, 1993.

14. Merriam, M.L., "An Efficient Advancing Front Algorithm for Delaunay Triangulations," AIAA Paper 91-0792, 1991.

15. Bowyer, A., "Computing Dirichlet Tesselations," The Computer J., Vol. 24, No. 2, pp 162-166, 1981.

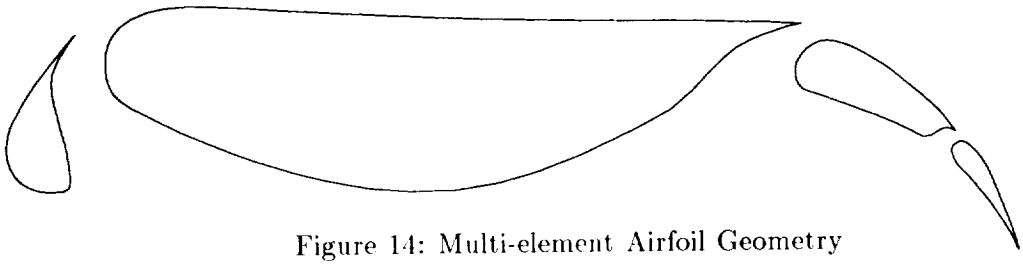


Figure 14: Multi-element Airfoil Geometry

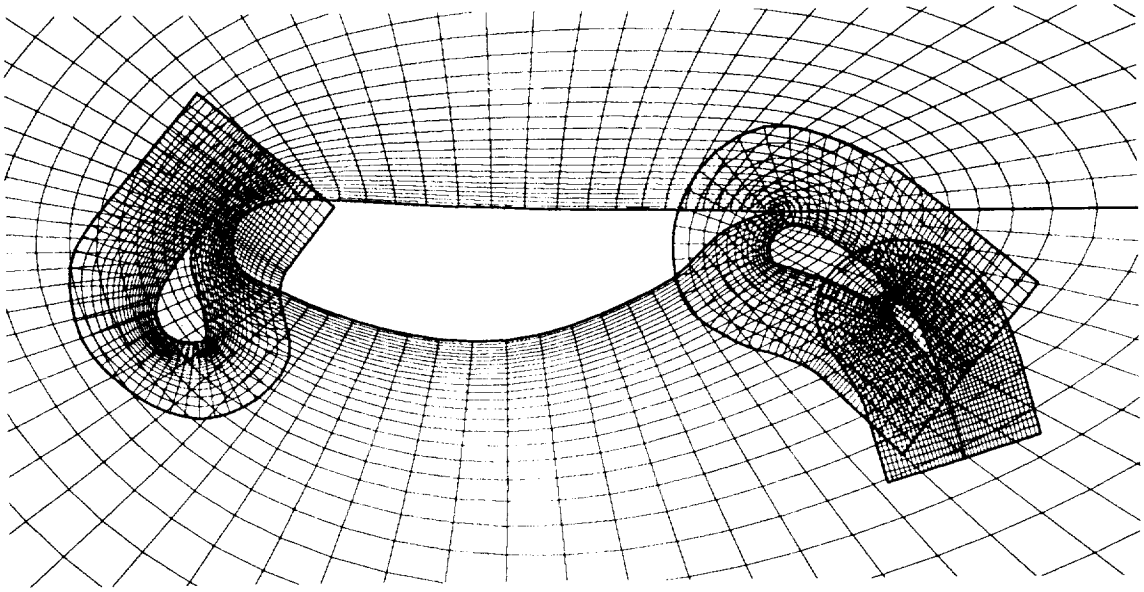


Figure 15: Set of Overlapping Structured Grids for Multi-element Airfoil Geometry

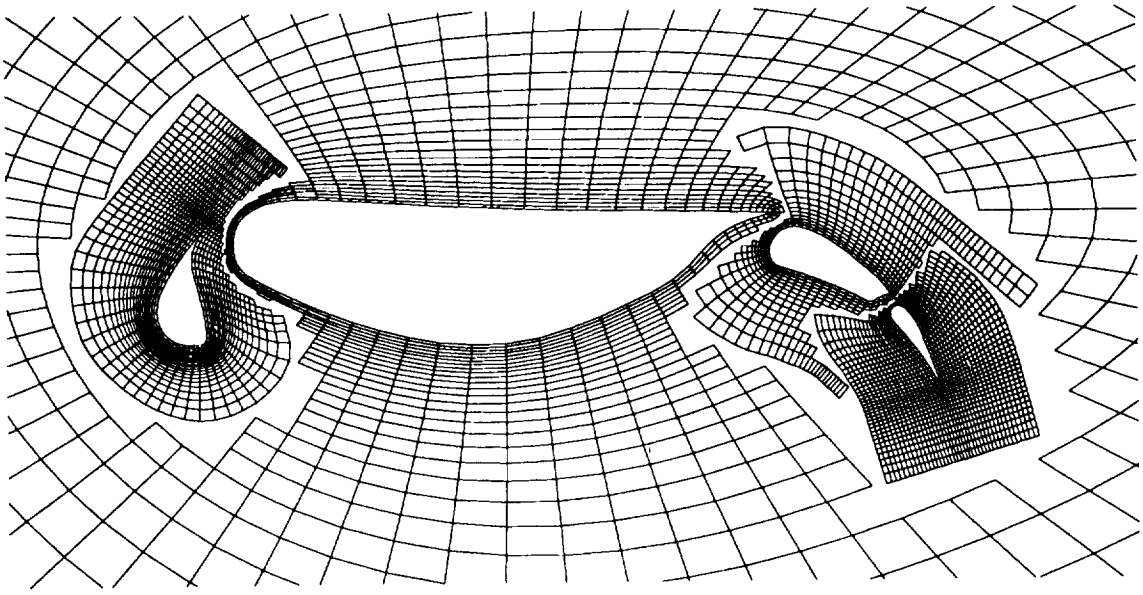


Figure 16: Structured Grids with Overlap removed for Multi-element Airfoil Geometry

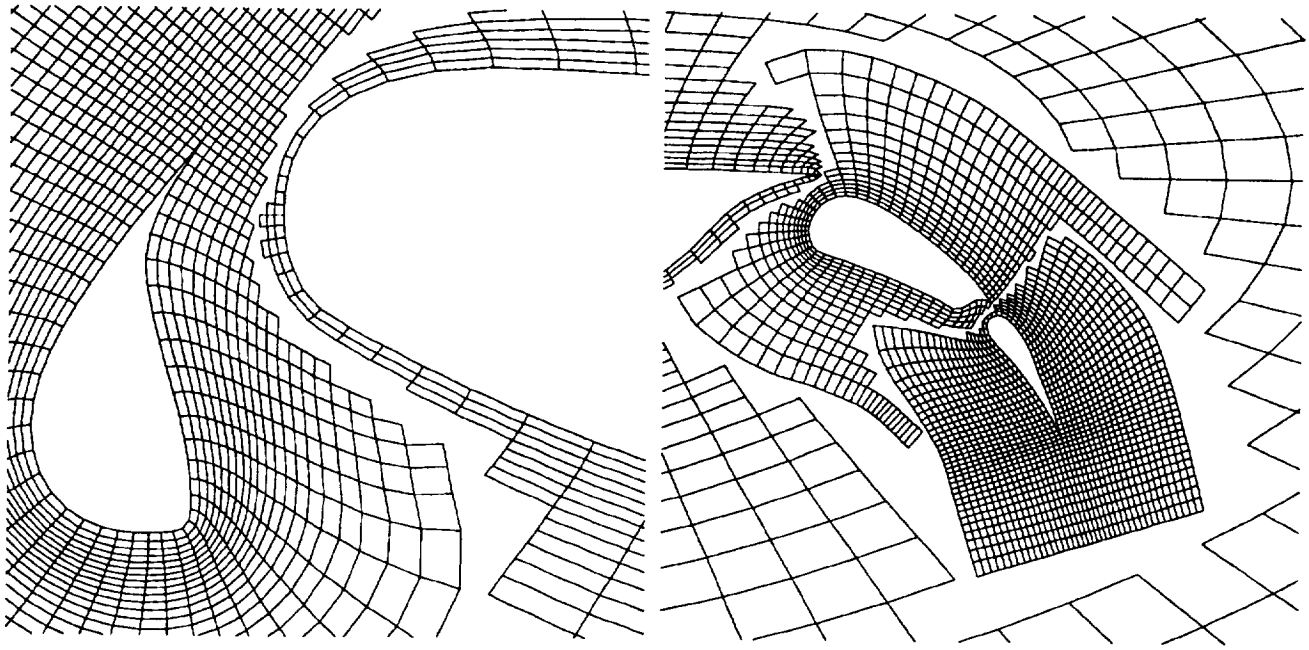


Figure 17: Closeup of Leading(l) and Trailing(r) Edge Flap Region for Structured Grids with Overlap removed

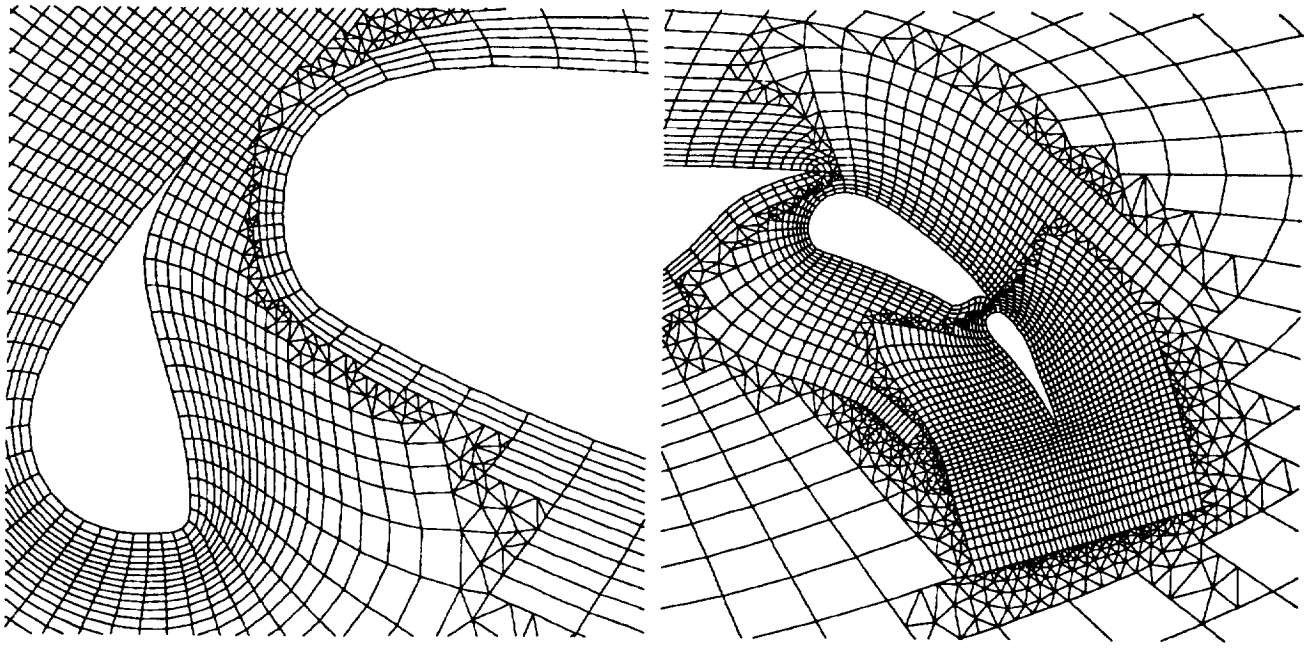


Figure 18: Closeup of Leading(l) and Trailing(r) Edge Flap Region of Hybrid Grid With Refinement of Front

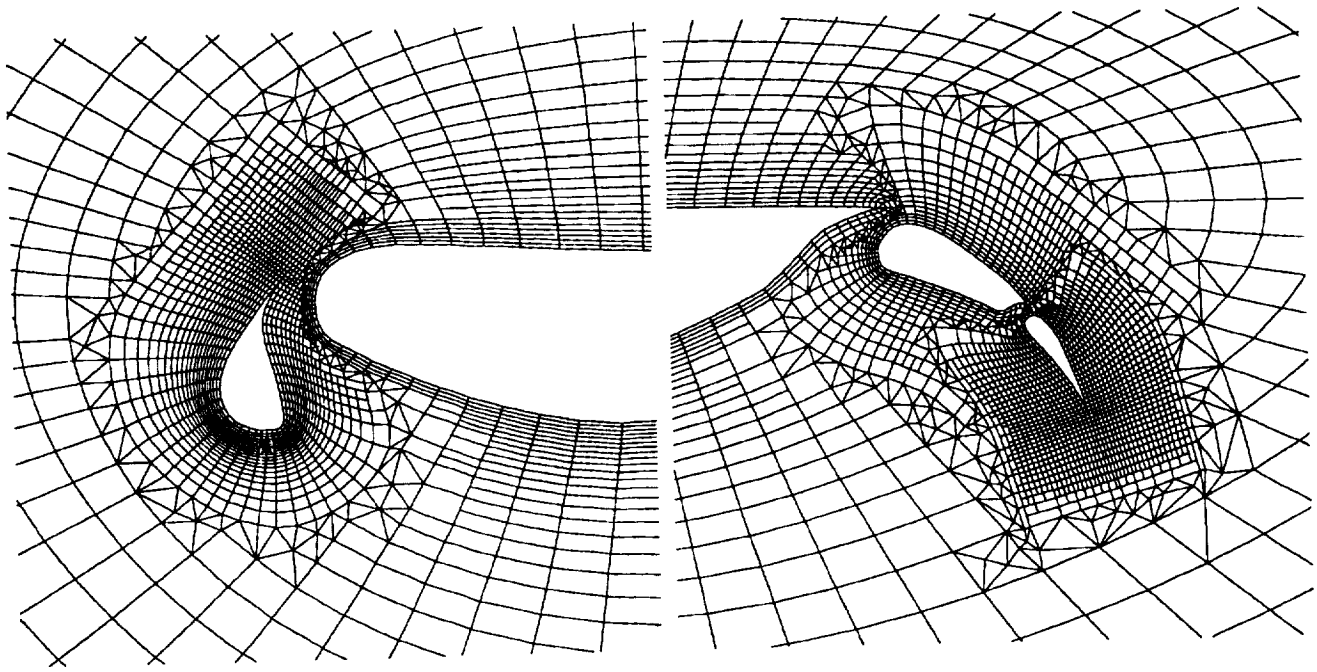


Figure 19: Closeup of Leading(l) and Trailing(r) Edge Flap Region of Hybrid Grid With Derefinement of Front

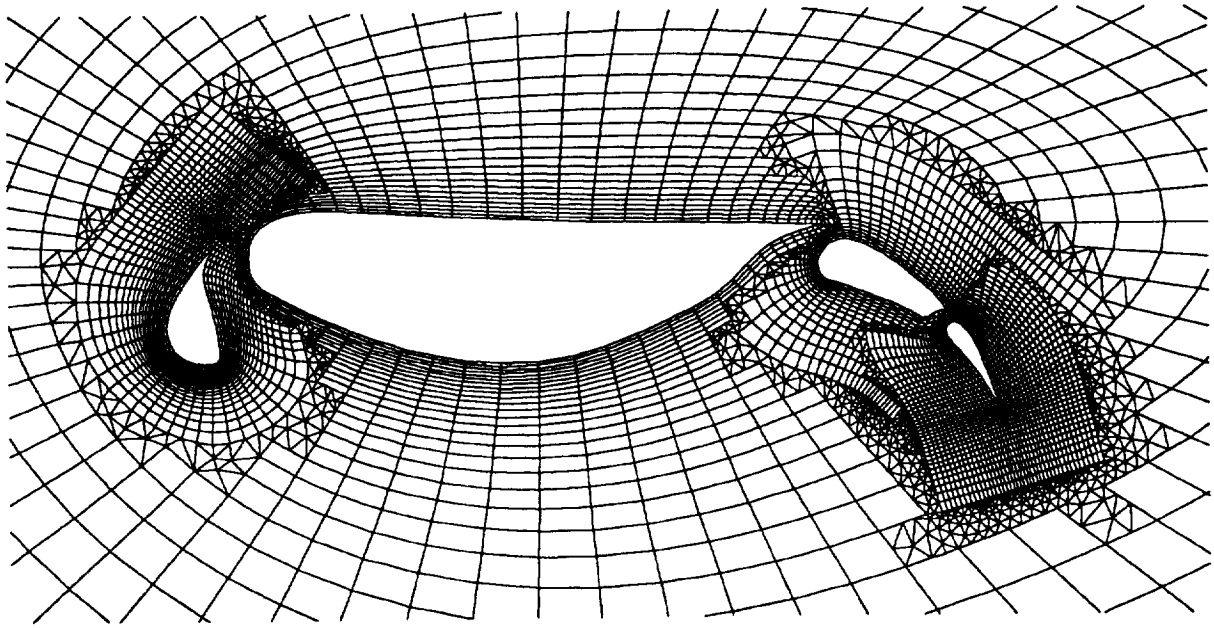


Figure 20: Final Hybrid Grid for Multi-Element Airfoil with Refinement of Front Between Structured and Unstructured Grid Regions

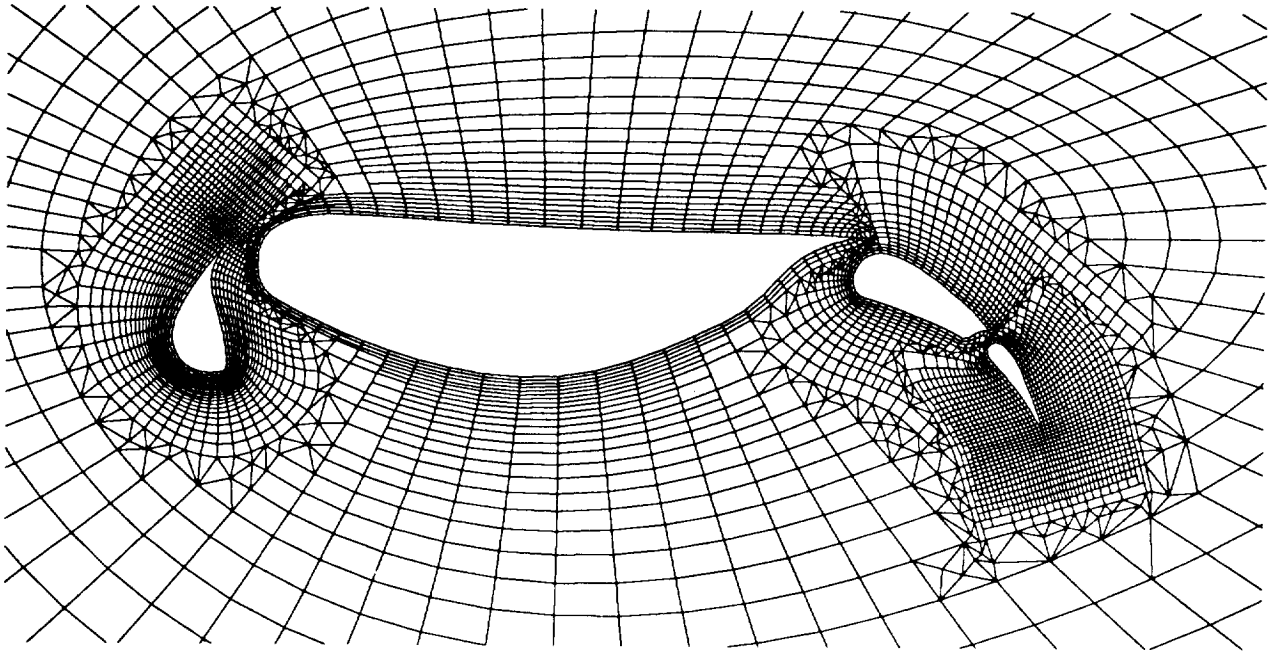


Figure 21: Final Hybrid Grid for Multi-Element Airfoil with Derefinement of Front Between Structured and Unstructured Grid Regions

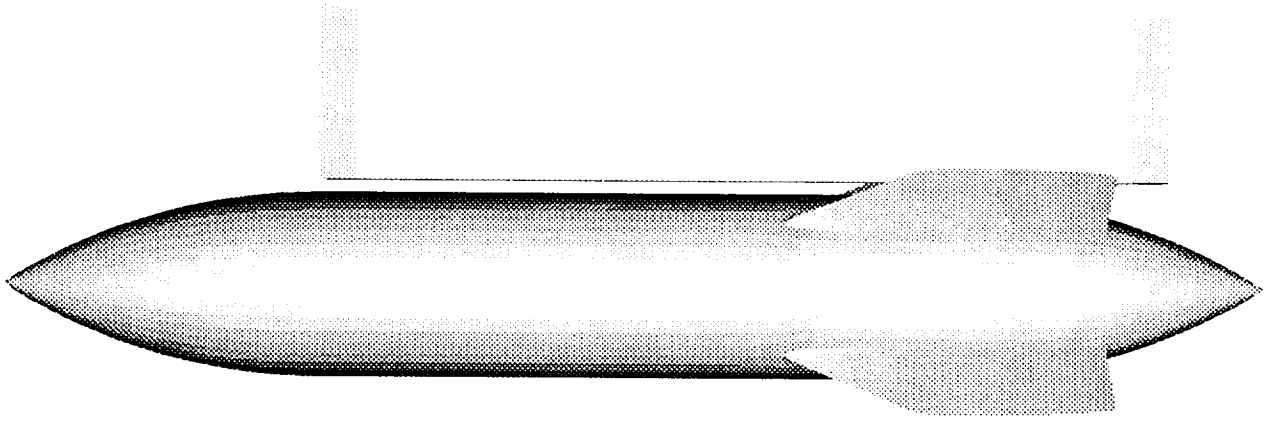


Figure 22: Pylon and Store Geometry

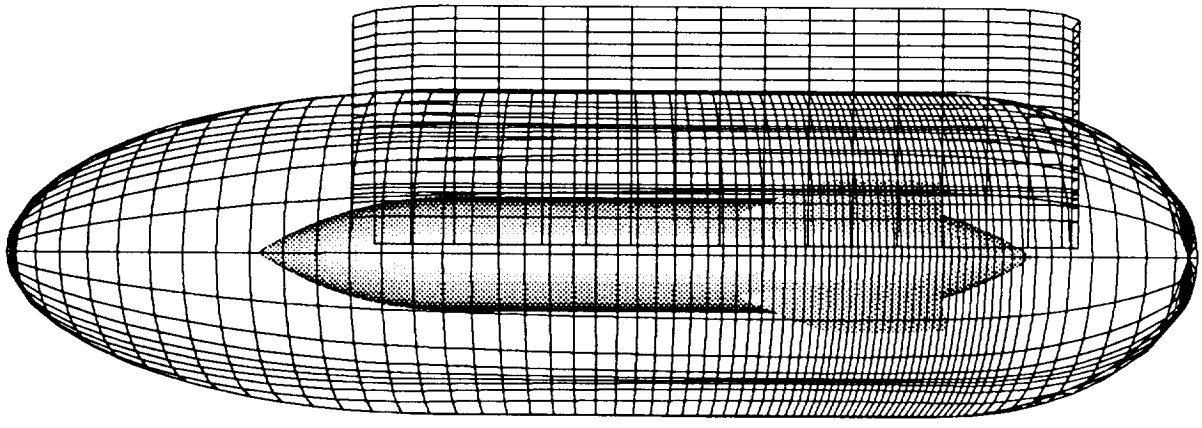


Figure 23: Pylon and Store with Portion of Grids Surrounding the Pylon and Store

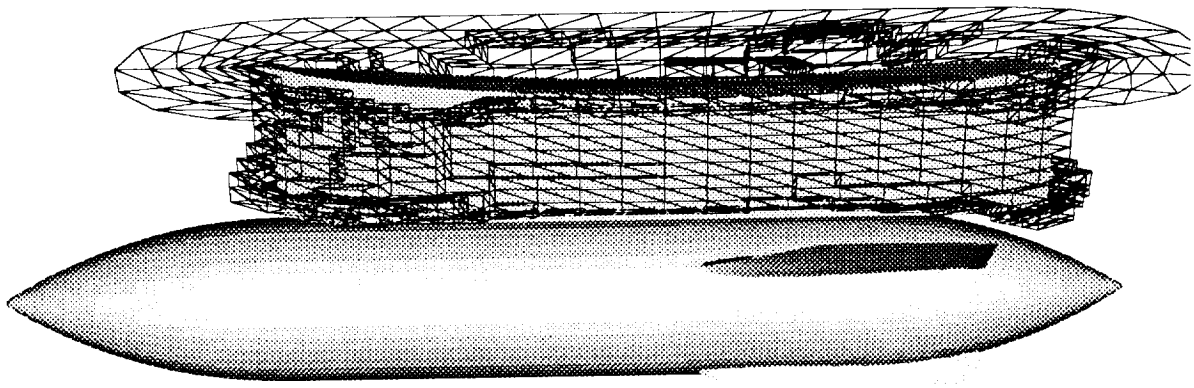


Figure 24: Store with Front from Removal of Overlapping Portion of the Pylon Mesh

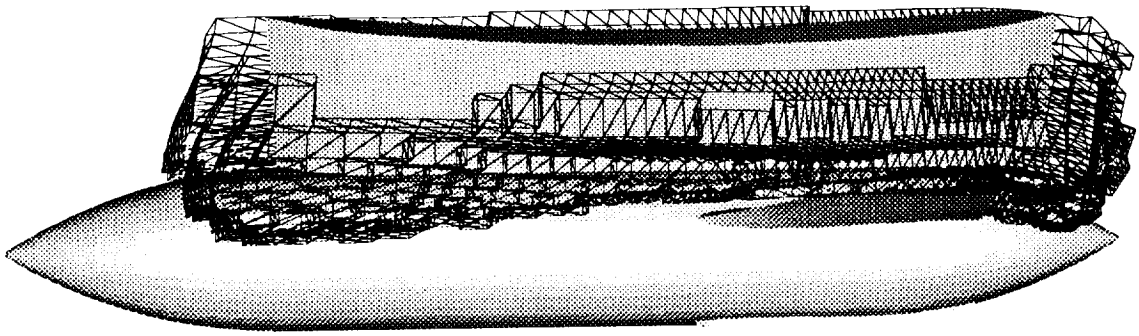


Figure 25: Store and Pylon with Front from Removal of Overlapping Portion of the Store Mesh

# TECHNOLOGY ASSESSMENT

N95- 28746

# EMERGING CFD TECHNOLOGIES AND AEROSPACE VEHICLE DESIGN

Michael J. Aftosmis
US Air Force Wright-Laboratory / NASA Ames
Moffett Field, California

## OVERVIEW

With the recent focus on the needs of design and applications CFD, research groups have begun to address the traditional bottlenecks of grid generation and surface modeling. Now, a host of emerging technologies promise to shortcut or dramatically simplify the simulation process. This paper discusses the current status of these emerging technologies. It will argue that some tools are already available which can have positive impact on portions of the design cycle. However, in most cases, these tools need to be integrated into specific engineering systems and process cycles to be used effectively. The rapidly maturing status of unstructured and Cartesian approaches for Inviscid simulations makes suggests the possibility of highly automated Euler-boundary layer simulations with application to loads estimation and even preliminary design. Similarly, technology is available to link block structured mesh generation algorithms with topology libraries to avoid tedious re-meshing of topologically similar configurations. Work in algorithmic based auto-blocking suggests that domain decomposition and point placement operations in multi-block mesh generation may be properly posed as problems in Computational Geometry, and following this approach may lead to robust algorithmic processes for automatic mesh generation.

## I. INTRODUCTION

Over the past 20 years, Computational Fluid Dynamics has made significant progress toward generating accurate simulations of flows around realistically complex aerospace configurations. While pundits are quick to point out that there exist multitudes of topologically simple model problems which quickly reveal shortcomings in turbulence models, dissipation models or advection schemes, a widening class of problems has moved within reach. Thus, although some regions of the flight envelope remain outside the realm of affordable and reliable numerical simulation, a growing body of evidence suggests that many critical situations may be predicted with accuracy. As a result of this increased confidence, the past decade has witnessed a shift in the focus of the CFD community from studying flow physics on topologically simple model problems toward ever more bold attempts at simulating vehicles in flight. This shift is evident throughout the military laboratories, NASA and industry as new codes are developed with increasing attention to generality and utility.

### I.A Computational Fluid Dynamics in Aerospace Design

From the first studies of numerical techniques for solving the Euler equations by Courant[1] Lax and Friedrichs[2], and the landmark calculations by MacCormack[3], CFD development has centered on issues of accurately solving the governing equations of fluid mechanics. This work set the tone for much of the subsequent development. Implicit schemes for centered spatial operators were presented in the mid 70's by Briley (1975)[4], and Beam and Warming (1976)[5]. Jameson *et al.*[6] introduced a very successful finite volume Runge-Kutta scheme in 1981 at about the same time that Enquist and Osher[7], Osher[8] and Roe[9],[10] were beginning development of approximate Riemann solvers which lead to many successful upwind methods in the years that followed.

This brief chronology highlights a major point when one considers CFD applied to the design cycle. While Steger had begun to consider complex configurations as early as 1978[11], development did not begin to concentrate on design or applications CFD before the mid 1980's. In 1985 Benek, Buning and Steger[12]



PRECEDING PAGE BLANK NOT FILMED

introduced a 3D chimera scheme for application to problems with realistic geometric complexity. This occurred at approximately the same time as other segments of the community pursued multi-block structured solvers and triangular mesh schemes for confronting the same issues[13],[14].

Euler[14] and Navier-Stokes[15],[16],[17] computations of flow around complete aircraft began to appear in the latter half of the 1980's. While these represented stunning achievements, they also served as omens which, it may be argued, the community was slow to identify and act upon. Grid systems for some of these early calculations consisted of single block meshes which literally took man-years to develop[15],[16],[17]. Multi-block approaches for Navier-Stokes simulations of complete configurations appeared in 1987[18]. However, while multi-block was a significant step forward, these efforts were also largely singular and did not directly focus on streamlining the grid generation or surface modeling steps in the process.

The evolving situation was documented in 1992 in an address by Cosner[19]. Figure 1 is taken from this reference and we use it here to epitomize the experience of designers and applications CFD groups. The figure depicts the breakdown of man-hours required to obtain two Navier-Stokes solutions of the transonic flow around an F/A-18 on meshes with about 3 million nodes. Even with the block structured grid generation tools available to industry in 1992, this chart shows that only about 14% of the man-hours were dedicated to running the solution. Fully 80% of the effort was spent on grid generation and geometry acquisition. Note that man-hours generally represent a direct monetary cost to the work's sponsor.

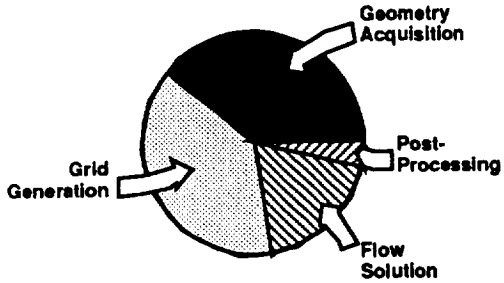



**Figure 1.** Breakdown of man-hours from CFD solution of F/A-18E/F configuration by McDonnell Douglas in 1992. (Reprinted from Ref.[19] with permission).

In the early 90's, examples like this lead to a better fundamental understanding of the importance of surface modeling and grid generation to the overall of the CFD solution process. However, excessive time for grid generation and surface modeling are not the only roadblocks to CFD's use in design and applied aerodynamics environments. In a well conceived article presented to the AIAA in 1991, Garner *et. al*[20]

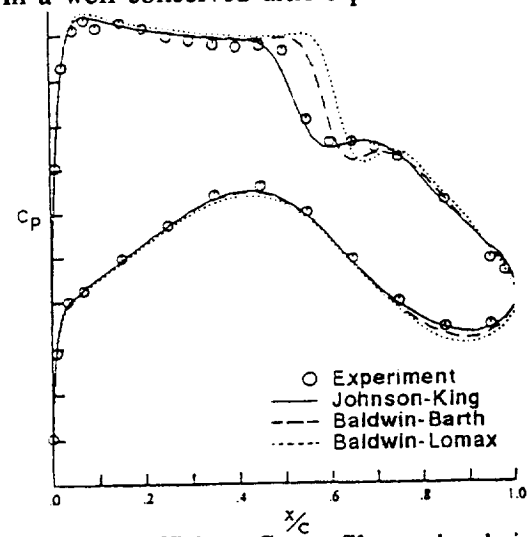



**Figure 2.** Sensitivity of surface pressure coefficient, $C_p$, profiles to the choice of turbulence model for a 3D transport wing. (Reprinted from Ref.[20] with permission).

highlighted several fundamental obstacles facing CFD's use within a design process. Figure 2 is an excerpt from this report. These data depict the result of three separate thin-layer Navier-Stokes solutions around a wing, varying only the turbulence model. The figure compares the distribution of the surface pressure coefficient, $C_p$, resulting from these calculations. In examining these figures, the designer becomes aware of the tremendous sensitivity of the shock location to turbulence model (nearly 10%$c$ variation over the 3 models). Subsequent improvements in the models may reduce this variation somewhat, but the essential problem of discrete solution sensitivity to turbulence model remains.

A similar situation is shown in Figure 3 - excerpted from the same report. This figure highlights the sensitivity of surface pressure to grid resolution at the trailing edge of the wing. These results emphasize that simply generating a mesh (even automatically) is not sufficient. The grid must accurately resolve not only the geometry, but also the physics of the flow. Of course, if a grid system takes weeks to setup, then designers cannot be expected to iterate through multiple cycles of grid generation and flow solution.

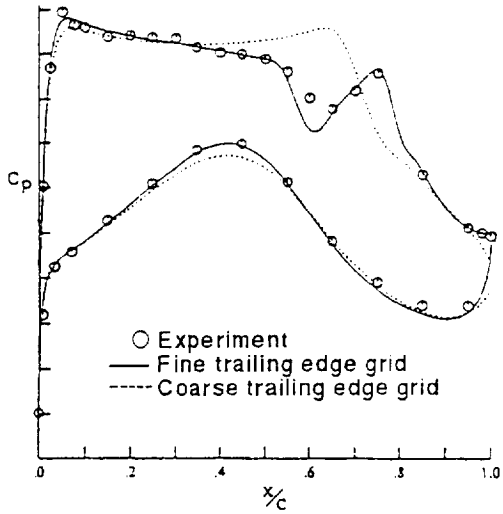



**Figure 3.** Effects of trailing edge mesh resolution on surface $C_p$ profiles for a 3D transport wing. (Reprinted from Ref.[20] with permission).

## I.B. Efficiency of the Numerical Simulation Process

The preceding three figures summarize the main difficulties historically associated with CFD in design; setup time, cycle time, physical modeling, and risks associated with inexpert generation of grids or setting of parameters within codes. These factors all raise the risks (and expense) associated with CFD's incorporation into aerospace design.

The aspect of time deserves special emphasis, both as measured in calendar time and in man-hours. For example, reducing the simulation time could allow multiple cycles to be run and therefore lower the risk involved in the process. Since this cycle is dominated by the human efforts of engineers and scientists, shortening the process generally equates to automation. Such increased automation opens the door for engineers with less specialized training. As a result, motivation exists on many levels to decrease simulation cycle times.

Other aspects of the impact of cycle time on design and applications CFD are less obvious from the perspective of a research environment. In a thoughtful review article presented to the AIAA in 1994, Rubbert[21] presents an economic model of the aircraft industry which further emphasizes the fundamental importance of time within the design cycle. This model is used to demonstrate that market share is inversely proportional to the time required to produce an aircraft of fixed specifications. Initially it seems intuitive that a company with a slow design cycle will loose market share to quicker, more flexible manufacturers of comparable products. However, the example documented in Figure 1 shows that the promise of reaping the

benefits of full CFD analysis in a design asks that manufacturers adopt a process which is 80% "overhead". Converting file formats of surface geometry files, computing lines of intersection between surfaces, laying-out grid block topologies, matching faces of grid blocks and stretching ratios of grid points, all represent, essentially, internal steps. They require highly skilled individuals to perform repetitive tasks on expensive equipment, and produce only an intermediate result. After they are used, the intricate details of the process add very little value, unless the experience acquired reduces the overhead of the next project.

Over the past five years, research groups have increasingly focused on increasing the efficiency of the simulation process. This work has revolved around building processes and systems which permit the utility of CFD to be limited by the accuracy of the physical models and numerical algorithms - rather than the time taken to re-grid a configuration when a pylon is moved, or when the aircraft's outer moldline must be modified due to structural considerations.

### I.C. Overview of Emerging Technologies

The techniques reviewed by this paper fall into several categories. *Unstructured* methods include any approach in which the tessellation of the computational domain or sub-domains does not map onto a logical right parallelepiped. This classification is therefore general enough to include methods with grids of pure tetrahedra, hybrid meshes, unstructured hexahedral meshes, and some implementations of Cartesian methods. Also included will be several methods which control or "drive" traditional multi-block grid generators and solvers. Such control based techniques generally proceed through direct automation of the tasks involved in the layout and construction of overset or abutted block structured meshes.

More specifically we will examine:

*Pure unstructured* – Tetrahedral based approaches for mesh generation and flow solution.

*Hybrid methods* – Schemes based on mixed elements, including prismatic/tetrahedral meshes intended to resolve viscous near-body layers on prismatic elements, and inviscid regions using tetrahedra (e.g. [22], [23], and [24] ).

*Cartesian methods* – Cartesian, non-body fitted methods, in which the geometry is "cut out" of subdivided Cartesian cells within the mesh. Algorithms for such tessellations may consider the mesh through unstructured, octtree structured, or embedded (and *i,j,k* structured) operations.

*Automation / Controllers for Block Structured CFD Schemes*

Auto-blocking – Automatic generation of sub-domains and required connectivity based on global information and surface geometry specifications.

Grid Abstractions – Topology of surface geometry is communicated to grid generator through building-block like geometric abstractions.

Scripting – Block layout and grid generation is handled automatically for pre-defined generic topologies. Operating system level controllers also run flow solution and automate post-processing of results (see the article by Buning[25] in these proceedings).

Macros – During grid generation, the user's actions are "recorded" in a macro, which may be played back to generate meshes around other geometries with the same topology.

A survey paper such as this is based upon a subjective view of the research it discusses. Inevitably, the author's opinions and experiences filter the material and flavor the discussion. Nevertheless, this paper attempts to present a valid perspective on the issues discussed, and concentrate on pertinent topics of general interest and concern.

## II. EMERGING TECHNOLOGIES

This section presents overviews and assessments of promising techniques which are aimed at reducing the time required for numerical simulation. Rather than attempt an exhaustive review of each topic, the discussion will lead through a general overview supported with important results and discussion stemming from analysis within the research community. Since these technologies are in various stages of maturity, an

attempt has been made to document the current status and summarize the potential outcome of future research. Subsequent sections will use this information to investigate ways in which these methods can be effectively used within the design and applications CFD environment.

## II.A. Pure Unstructured Methods

Techniques involving unstructured grids composed of triangles (in 2D) or tetrahedra (in 3D) have been applied to the Euler and Navier-Stokes equations since the mid- 1980's.[26] Based upon theoretical foundations developed largely in the late 1970's[27],[6], such methods have been attractive from the outset since their spatial discretization operators avoid unnecessary assumptions about global mesh topology. Thus, solvers view the mesh as an unstructured collection of cells, with some explicitly defined connectivity. The grid generation task reduces to the more general problem of generating and tessellating coordinate data between prescribed boundaries, and is therefore more amenable to automation. This section will detail the current state of such research and discuss approaches for flow simulation and mesh generation for inviscid and viscous simulations.

### II.A.1 Unstructured Flow Solvers

Unstructured simulations using central difference finite volume techniques were applied to the inviscid simulation of a complete aircraft as early as 1986[26]. The short span of time from inception to application on such complex configurations provided significant motivation for their continued development. Several outstanding reviews and summaries exist to track development of these methods over the past decade. Excellent articles have been prepared by Venkatakrishnan[28] and Mavriplis[29] chronicling progress in both flow solver development and mesh generation. In addition, the proceedings in reference [30] provide a detailed description of a variety of unstructured methods with supporting theory and applications, including domain decomposition for parallel architectures, implicit time integration, and a host of mesh generation strategies.

The first flow solvers of Jameson and Mavriplis [26] operated in a cell-centered finite volume environment. In the subsequent 3D work, which appeared in 1986, Jameson *et al.* [14] extended these ideas to permit vertex based formulations and related the central difference finite volume framework to a Galerkin finite element approach with linear basis functions. Desideri and Dervieux[31] presented one of the first higher-order accurate upwind schemes for triangular meshes in 1988. This work relied upon van Leer's MUSCL[32] scheme to construct a node-based upwind method on meshes composed of simplices. Since unstructured meshes have no dominant coordinate direction, some ambiguity existed in the implementation of one-dimensional limiting ideas and the use of directional operator splitting on unstructured meshes[33]. This shortcoming was overcome when Barth and Jespersen[34] presented an unstructured upwind method based on a central difference gradient estimation that obeyed multi-directional monotonicity principles.

In the past five years, a host of improvements have been proposed to many of these algorithms. Just as importantly, however, theoretical frameworks have been introduced which give insight into commonalty between various schemes and aide in developing efficient implementations. For example, reference [35] proposed the use of edge-based formulas and data structures which have since become one of the major contributions of the research. Since that time, a variety of authors have invoked minimal storage and computational complexity arguments to further support the use of such structures in node based schemes [35],[36],[37].

Most unstructured Navier-Stokes work has adopted either central difference finite volume or finite element discretizations of the diffusive terms in the governing equations (see for ex. [35], [38], [39], [40], among others). Like the convective terms, these operators are also amenable to edge-based data structures which permit compact construction and storage. Reference [28] provides additional sources of discussion concerning recent developments in unstructured Navier-Stokes solvers.

### II.A.2 Mesh Generation

Initial efforts with unstructured techniques generally relied upon meshes constructed from quadrilateral structured meshes through the addition of a diagonal spanning each cell[26]. In their initial complete aircraft calculations, however, Jameson *et al.*[14] began to invoke principles of Delaunay triangulation in tessellating point data and related techniques have become commonplace in subsequent mesh generation strategies. This initial work identifies the main strength of unstructured approaches - that is, the ability to automate generation of a volume mesh from an initial triangulation of the domain boundaries. Thus, the tedious, ambiguous, and often complex process of decomposing the computational domain into regular hexahedral blocks is avoided.

A Delaunay triangulation of coordinate data is one in which the circumcircle through the vertices of each triangle does not contain any other vertex in the mesh[41]. Most approaches can be classified as either Delaunay, advancing front[42],[43], or a combination of the two[44],[45],[46]. Recent research in this field has produced a variety of *Steiner Triangulation* techniques[47], in which new sites are incrementally inserted into existing, usually Delaunay, triangulations[48],[49],[50]. Such methods appeal to the unique properties of Delaunay triangulations[51] to produce meshes with mathematically provable control over maximum angles and mesh quality. For example, a Delaunay triangulation of a planar point data satisfies the min-max criterion, *i.e.* it leads to a unique graph which minimizes the maximum angle of any triangle in the mesh. Exploitation of this property, taken in conjunction with a circumcenter site insertion strategy permits advocates of Steiner triangulation techniques to produce triangulations which guarantee that all angles in the final mesh will lie between prescribed minima and maxima[52].

The phenomenal success of references [42], [44], [53], and others, attests to the fact that two and three dimensional meshes of isotropic triangular or tetrahedral elements can be robustly generated with well documented methods. Current research is therefore directed at generation meshes containing stretched elements in a similarly reliable manner[46],[49],[50],[54]. Although many techniques have been proposed, this topic remains one of the pacing items in the application of unstructured Navier-Stokes solvers to complex configurations.

### II.A.3 Current Status

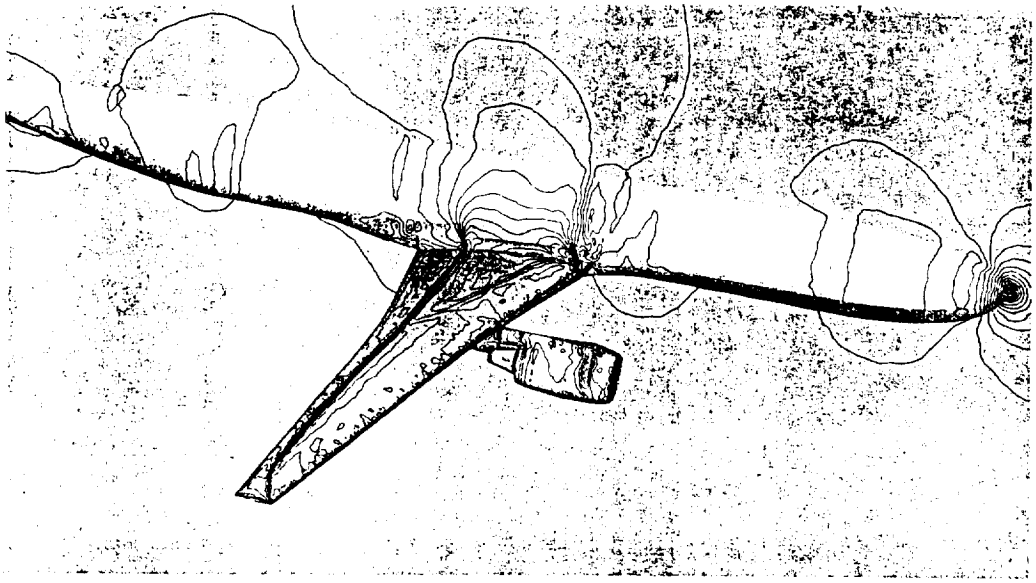
The recent literature has witnessed a virtual explosion of research in unstructured methods. However, looking with an emphasis on design and applications CFD, we are able to consider a smaller subset of this work which is restricted to 3D applications. Within this subset, several classifications still exist. Research pursuing efficient and accurate Navier-Stokes solution represents a large body of work and several competing methods are being debated. Nevertheless, applications are not yet commonplace, and one may argue that this work is still formative. Similarly, despite some impressive results[55],[56],[57], time dependent adaptive schemes and re-meshing algorithms such as those for unsteady flows and moving body problems, is also largely a research topic. Excluding these topics we are left with work which presents applications of unstructured methods to 3D steady flow fields.

This process of narrowing down and classifying ongoing research is instructive. It emphasizes the point that although the research community exerts considerable resources in unstructured flow solvers and mesh generation, only a small fraction of this effort is dedicated to working problems directly related to the generic application of these methods. Relatively few researchers focus directly on issues surrounding efficient implementation and routine application. Such work is, perhaps, almost unattractive in an environment which rewards novelty and fundamental concepts. However, from an industrial or applications CFD point of view, the automation of routine processes is of paramount importance, and in this polarity lies a serious conundrum.

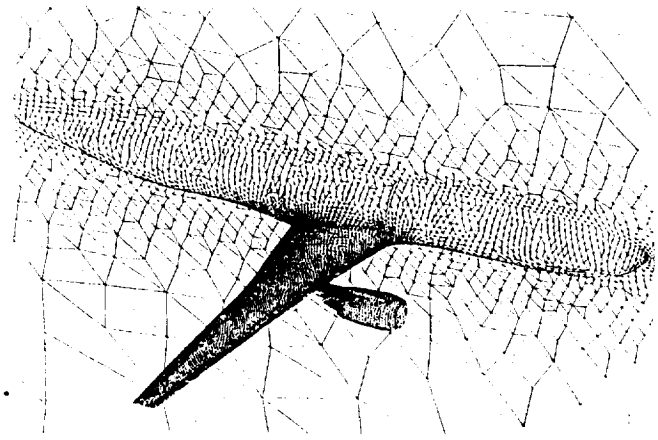
Within the subset of steady, inviscid applications, huge progress has been made since the first complete aircraft simulations of reference [14]. Good examples of relatively efficient codes may be found

throughout the recent literature[40],[43],[37],[58]. Figure 4 contains representative results from this body of work, and was taken from the results of Mavriplis and Venkatakrishnan [58]. The figure depicts a Low Wing Transport (LWT) configuration with pylons and nacelles computed on a mesh of 804,056 vertices connected by a graph that forms 4.5M tetrahedra. The flow is transonic at Mach 0.77 and an angle of attack of 1.116°. This is a good example for the present discussion, since the code was written to take advantage of edge-based data structures, and requires only approximately 150 words of storage per vertex (relatively few for an unstructured multigrid code). Convergence acceleration of the multi-stage Runge-Kutta time stepping scheme is performed through agglomeration multigrid[58] which generates coarser meshes by fusing together fine grid cells. This multigrid strategy is particularly well suited to complex geometry, since it avoids the traditional shortfall of failing to resolve the geometry on coarser meshes in the grid sequence.

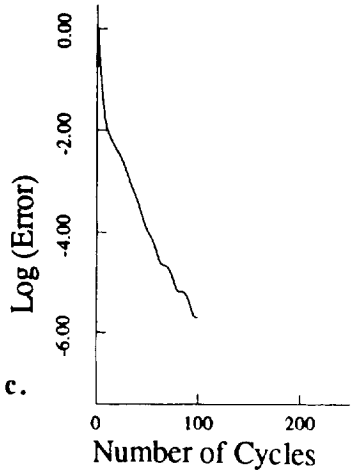
Figure 4a shows a view of the surface grid for the first coarser mesh in the multigrid cycle (containing 106,064 nodes). Figure 4b contains surface and symmetry plane Mach contours showing the lambda shock structure on the wing, and the effects of propulsion integration. The convergence history depicted in Figure 4c indicates that the $L_2$ norm of the residual was reduced six orders of magnitude in 100 multigrid cycles.



4b.



4a.




4c.

**Figure 4.** (a) Surface mesh for coarse grid with 106,064 vertices, fine mesh (not shown) contains 804,056 vertices. (b) Surface and symmetry plane Mach contours for transonic flow over LWT configuration. (c) Convergence history of 7-level agglomerated multigrid for LWT case. (Frames reprinted from reference [58] with permission).

The simulation required 96MW of memory and 90 minutes of single processor CPU on a Cray Y-MP. These results are representative of the CPU and memory requirements of modern, well-written unstructured solvers.

II.A.4 Unresolved Issues and On-going Research

Surface modeling – In the context of unstructured methods, surface modeling plays two important roles. First, an initial triangulation (which is constrained to the surface geometry) must be supplied to the volume mesher for generation of the initial mesh. Secondly, if flow field driven adaptation is permitted, new sites may be inserted on the boundary and these too must conform to the actual surface geometry. Generation of such a constrained initial surface triangulation is non-trivial when one considers that the surface may be originally described through CAD data, loftings, natural NURBS, trimmed NURBS, patches or surface grids. Edges in this triangulation must follow leading edges, trailing edges, junctures between components, and may not be allowed to swap (if edge swapping will produce faces which no longer coincide with the surface description). Generation of this surface triangulation is typically the most time consuming and user-intensive operation in the mesh generation procedure. While volume meshing may take minutes, or tens of minutes on an engineering workstation, the initial triangulation may take hours to days. Moreover, if the surface triangulation is computed with a preprocessor or commercial software package, the true geometry description may be unavailable for later use, and adaptation becomes a clumsy process.

Anisotropic Mesh Generation – Several approaches have been proposed for creating meshes with high aspect elements suitable for Navier-Stokes including those in references [54], [59], [46], and [50] among others. However, research involving the retention of angular control while still retaining a sufficiently smooth blending of anisotropic viscous cells with isotropic Euler elements continues. In addition, the insertion of high aspect ratio tetrahedra into general unstructured meshes increases the likelihood of generating "sliver" elements, non-tetrahedralizable regions, etc. Such effects have a detrimental effect on the robustness of viscous grid generators. Finally, even with the addition of solution based adaptation, difficulties exist concerning meshing wakes and other free shear layers with sufficient resolution[54].

Counting Arguments – A variety of authors have raised concerns about the total numbers of cells and edges required to fill volume space with high aspect ratio tetrahedra. As an example, suppose we seek to grid the boundary layer region around a single element 3D wing at elevated Reynolds number. A structured surface mesh may contain 200 vertices around the chord and perhaps 50 spanwise stations. Spanning an expected attached boundary layer with 25 stations would permit sites at $y^+$ of around unity with reasonable grid stretching. Thus the boundary layer portion of the mesh would consist of $200\text{x}50\text{x}25 = N = 250{,}000$ vertices, and hexahedral cells made up of 750,000 edges (assuming we remain away from mesh boundaries). Tessellating this structured region with right tetrahedra would lead to dissecting each hexahedral cell into $\alpha = 5$ tetrahedra (best case). This leads to 1.25M tetrahedrals and $(\alpha+1)N = 1.5$M edges. Thus, for an explicit, edge-based, code we have doubled the storage and *per sweep* CPU requirements simply by virtue of the tessellation. In an implicit code the edges must still be stored, but now the additional connectivity implies a fuller matrix for inversion. Moreover, recent analysis and numerical experiments suggest that for certain classes of schemes, the additional diagonal edges do not necessarily lead to greater accuracy[60]. (In fact, Ref. [61] has actually demonstrated improved discrete solutions through choosing control volumes which de-emphasize the contributions of such diagonal edges.)

## II.B Hybrid Methods

The counting arguments, accuracy concerns and other issues in the preceding paragraphs have prompted several groups to consider alternatives to pure unstructured methods [22],[23],[24],[62],[63]. For example, reference [60] suggested that diagonal edges found in semi-structured viscous layers of triangular or tetrahedral meshes could be deleted to leave triangular, hexahedral or prismatic elements. Such edges incur a memory penalty with no apparent accuracy advantage and thus degrade the efficiency of the method. The

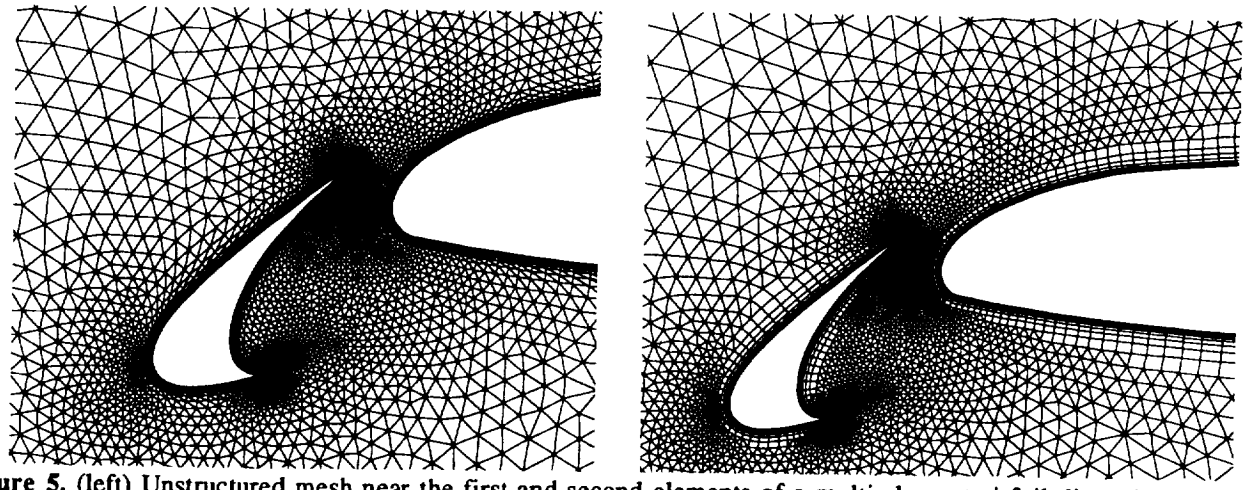


**Figure 5.** (left) Unstructured mesh near the first and second elements of a multi-element airfoil discretization. (right) Mixed element mesh where 11,544 triangles were collapsed to form 5,772 quadrilateral elements. (Frames reprinted from reference [46] with permission).

extension of edge-based unstructured flow solvers to accept hybrid elements is relatively straight forward and the idea has been investigated by several authors. Figure 5 contains two figures excerpted from the work of Marcum[46] and shows just such an approach. Sites forming triangles in the semi-structured wall boundary region near the leading edge of a muti-element airfoil have been re-tessellated as quadrilateral elements. In this example, 11,544 triangles near the wall were collapsed to form 5,772 quadrilaterals through removal of the unnecessary diagonals. In addition, reference [46] has investigated collapsing tetrahedral elements to form prisms in semi-structured regions of the mesh, but final results have not yet been presented.

II.B.1 Prismatic and Hybrid Meshes

The use of prismatic elements for efficiently discretizing viscous layers, or even entire domains has received increased attention since the late 1980's. Prismatic mesh generation ideas have been presented by references [23], [24] and [62] among others. The methods proposed by Nakahashi[23] and Kallinderis[62] are based on algebraic construction principles that begin with a surface triangulation of the geometry. Families of mesh lines spawn from mesh vertices in the surface triangulation. The mesh is constructed layer by layer in a manner which is analogous to an inflation of the surface triangulation. This construction technique permits a semi-structured interpretation of the mesh. The lines emanating from vertices on the body are continuous, and nodes along each of these are vertices of fixed degree. This local structure permits straight forward implementations of semi-implicit multigrid, or even semi-coarsened multigrid convergence acceleration[24],[64].

The algebraic marching procedure described here clearly has the character of a hyperbolic algorithm for the positioning of sites on subsequent layers of the mesh. Recognizing this, Pandya[24] has proposed alternative construction schemes based on solving hyperbolic partial differential equations for determining the vertex locations on successive mesh layers.

One criticism of early prismatic mesh generation techniques was that the extension to multiple body configurations was unclear, since there appears to be no clearly defined manner in which to prevent multiple prismatic meshes to fuse together as they grow. To address this issue, recent work presented by Kallinderis[22] proposed constructing layers of prismatic elements only near the wall boundaries to resolve viscous layers, and then discretizting the rest of the domain using an advancing front technique, filling the space with isotropic tetrahedra. If prismatic layers from multiple elements overlap, then they are locally "receded", and the space in between may be filled with tetrahedral elements during the advancing front step. Reference [22] presents a variety of viscous multi-body meshes constructed following this procedure.

II.B.2 Current Status and Unresolved Issues

As compared with unstructured tetrahedral techniques, research into prismatic and other hybrid methods is relatively new. As a result, a smaller body of work exists to highlight their strengths or document potential shortfalls. Research in these methods has shown great promise, however there are still relatively few detailed computations on sufficiently fine grids.

Hybrid methods were developed in direct response to the concerns about anisotropic tetrahedra and counting arguments documented above, and they have successfully overcome many these obstacles. The research community has been quick to respond to criticisms concerning multi-body and meshing concave domains and is similarly working to resolve most outstanding issues. A partial list of potential roadblocks includes:

Surface modeling – Like pure unstructured methods, prismatic techniques and most other hybrid methods generally employ a surface triangulation to begin the mesh generation process. Thus, such methods inherit the difficulties associated with generation of such a triangulation that were presented, above, in section II.A.4

Adaptation - In hybrid techniques that rely upon the semi-structured nature of prismatic regions, the incorporation of adaptive mesh algorithms can become unclear. Parthasarathy *et al.*[64] have presented some promising work addressing this topic in their consideration of viscous flow around a sphere. In this work, the sphere was encased in a prismatic mesh to capture viscous phenomena and a tetrahedral mesh was employed away from the body. The hybrid adaptation scheme included *h*-refinement and mesh re-distribution. Nevertheless, the flexibility of mesh re-distribution algorithms on more general topologies remains an open question. Additionally, if a tetrahedral cell at the prism-tet interface is *h*-refined the entire stack of prismatic elements between this cell and the body must also be *h*-refined if one is to preserve the semi-structured nature of the prismatic layers. Such a strategy raises efficiency concerns. Of course, if the solver does not make use of the structure in the prismatic layers, and considers the mesh simply as an unstructured collection of mixed element types, the restrictions on adaptation largely disappear. Finally, we note that since prismatic meshes have an *O-O* topology, mesh adaptation to resolve wakes and/or free shear layers within the flow may present a problem

## II.C Cartesian Mesh Methods

Cartesian approaches differ from those presented in the previous sections by virtue of the fact that they make use of a *non*-body fitted mesh system to discretize the computational domain. The hexahedral cells in the domain are a set of right parallelepipeds and the original grid system may extend through solid wall boundaries in the computational domain. The process then removes any fully internal cells, flags cells which intersect the body and treats remaining cells as general volume mesh control volumes. The promise of the technique stems from the fact that it trades the case specific problem of generating a body fitted mesh (unstructured, structured, blocked, etc.) with a more general problem of computing and characterizing geometric intersections between Cartesian flow field cells and the surface geometry. Thus, all difficulties associated with meshing a particular geometry are restricted to a lower order manifold which constitutes the outer shell of the geometry - rather than occurring throughout the computational domain. Recent research has demonstrated that these mesh generation operations readily lend themselves toward automation and the method can be applied to extremely complex geometries[65],[66],[67],[68],[69]. Since solid wall boundaries may cut arbitrarily through the layer of "cut cells" encasing the body, surface boundary conditions are of obvious importance in Cartesian schemes.

Cartesian approaches fall into two general categories. Either they consider the mesh to be an unstructured (or octree structured) collection of *h*-refined hexahedra, or they operate by embedding structured sub-grids within existing structured blocks. In unstructured or octree methods, volume meshing of the computational domain relies on a simple and robust procedure of cell division. Beginning with a coarse background grid,

or even a single root cell, the hexahedral elements are recursively subdivided in order to resolve the geometry and evolving flow features. The final mesh is a collection of cells at various levels of refinement, viewed as either entirely unstructured or as having an underlying octree connectivity. Structured approaches embed *i,j,k* structured sub-grids similarly aimed at resolving geometric and flow field features. Successful 2D and 3D solution procedures have been proposed following both implementations[65],[67],[70],[71].

II.C.1 Development of Cartesian Approaches

Although three dimensional applications to complex geometry have only recently become commonplace, Cartesian approaches have been evaluated since the late 1970's. Work by Purvis and Burkhalter[72] solved the full potential equation on 2D Cartesian meshes using a finite volume method. Solution of the Euler equations was pursued in the mid-1980's by a variety of researchers[73],[74], and the first three dimensional inviscid solutions appeared in the late 1980's by Gaffney, Hassan and Salas[75].

Cartesian approaches have been successfully utilized in industrial applications including Boeing's TRANAIR code which solves the full potential equation, and the commercially available MGAERO[76] package for Euler simulations. These applications are notable because they provide close links with surface modeling and provide a wide base of experience with large-scale computations using Cartesian methods. MGAERO, for example, adopts a component based approach to complex geometries[77],[78]. Similar approaches have been adopted within the research documented in References [66],[67] and [79].

The cut cells necessarily present in Cartesian discretizations present unique problems in the implementation of accurate boundary conditions. References [80],[81],[82] and others present insight into the important issues involved.

The isotropic elements stemming from *h*-refinement of Cartesian hexahedra are well suited to resolving flow structures in inviscid simulations. However, use of such elements to capture boundary layers and other stiff viscous phenomena would be grossly inefficient[83],[84]. Recently, a variety of authors have proposed alternate techniques for extending inviscid Cartesian approaches to viscous flow[68],[84]. Historically, the lack of a clear extension to viscous simulations has been a weak link and the recently proposed venues offer the possibility of further research.

II.C.2 Current Status

In the recent literature, one finds a multitude of work stemming from Cartesian mesh methods. Many notable 3D computations have investigated the flow around complex geometries. The 3D Reynolds averaged Navier-Stokes results which use prismatic meshes near the body in reference [68] are particularly promising. However, such examples are still formative, and the technology has not spread across the community for independent confirmation. The target of most Cartesian work is a role within the inviscid design/analysis cycle. Appropriately, this section restricts its attention to the discussion of 3D Euler applications similar to those found in [67],[69],[78].

While unstructured and structured CFD approaches both involve surface modeling issues, the static nature of their surface description permits these techniques to develop sufficient surface meshes in preparatory steps. A notable exception, of course, are unstructured adaptive algorithms which must interrogate the surface database during each mesh adaptation phase. For similar reasons, Cartesian approaches demand that the mesh generation and adaptation algorithms be closely coupled with the geometric database. Throughout the mesh generation process, Cartesian cells must be constantly tested against the surface database. Again during adaptation, new cut cells are clipped against the surface, and fully internal cells are eliminated. In reference [66], tests were performed with direct inquiries to a NURBS description of the body. However, speed, robustness and other issues motivated a return to the use of a surface triangulation database in later research. Important differences exist between the surface triangulations required for Cartesian and fully unstructured or prismatic approaches. In Cartesian approaches, individual components of the geometry may be independently triangulated into closed polyhedra, without regard to overlap or intersection of the

component triangulations. Thus, the technique avoids computation of surface-surface intersections, and since Cartesian cells may cut arbitrarily through the geometry, they may intersect the triangulations of several different components without special handling. In essence, the actual *topology* of the full configuration need never be communicated to the grid generation routines, and herein lies the fundamental difference with structured of unstructured techniques. This single factor is perhaps the key in understanding Cartesian claims for rapid turn-around and automation. Within many mesh generation codes, it is relatively easy to obtain surface grids or triangulations for individual components if one does not worry about intersections with other parts of the overall geometry. As a result, the geometric modeling is substantially quicker for many applications.

Figure 6 contains an adapted Cartesian mesh generated by Aftosmis and reprinted from Reference [67]. The domain includes over 2.9M Cartesian cells and the surface geometry is described by 12 separate component triangulations. The final adapted grid contains 10 levels of cells and approximately half of these are at the finest level of refinement. The discrete solution began on an initial, geometry adapted, mesh with 5 levels of cells. Further cell refinement was based on the evolving flow solution. Figure 7 shows a close-up of the adapted mesh painted with isobars of the discrete solution on a plane located between the fuselage and first nacelle.

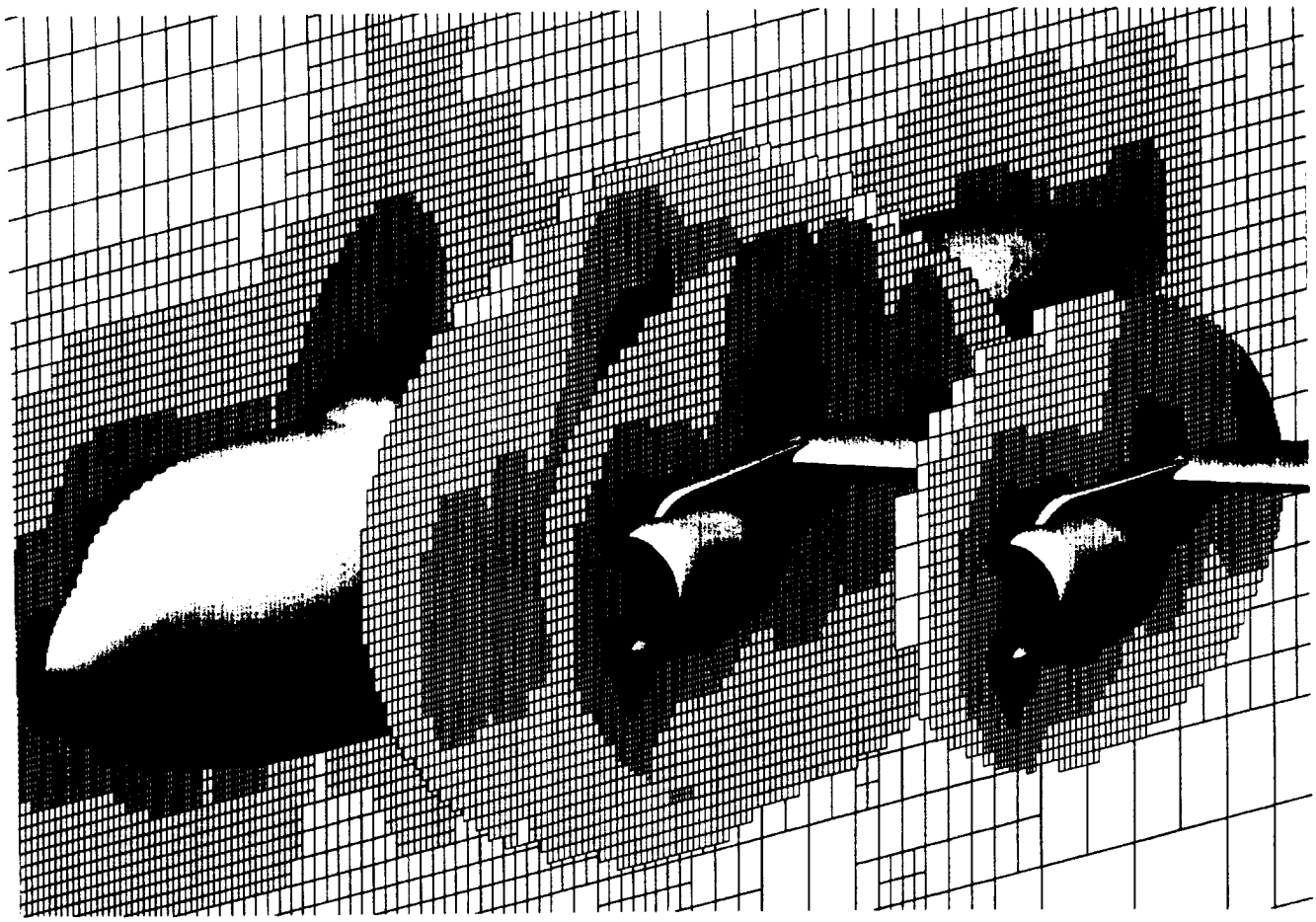


**Figure 6.** Adapted Cartesian mesh for High Wing Transport (HWT) configuration with 2.9M Cartesian cells and 10 levels of refinement. The surface geometry is described by 12 separate component triangulations. (Reprinted from Reference [67])

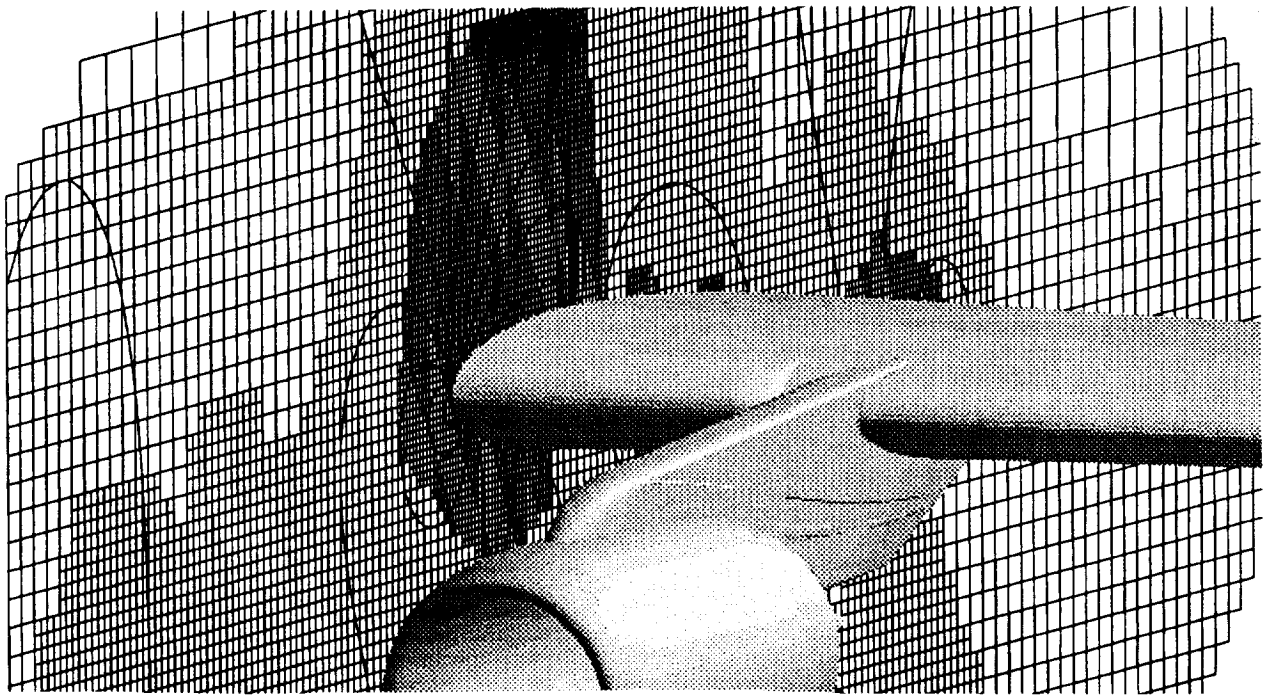


**Figure 7.** Close-up of the adapted mesh and isobars in the discrete solution on a plane located between the fuselage and first nacelle. (Reprinted from Reference [67])

In an effort to more clearly distinguish automated from interactive tasks in obtaining such solutions, it is useful to trace the primary steps in the process. The example used here is similar to the HWT configuration shown in figures 6 and 7, but with the addition of a winglet, and high lift devices (inc. leading edge slat, flap, flap vane, and spoiler). Figure 8 summarizes the contributions of the various steps as a fraction of the total time required for performing the complete simulation and analyzing the results.

CAD (18%) – Perform CAD operations to generate trimmed NURBS surface description of each component in configuration. *Interactive*

Surface Grid (18%) – Fix problems with NURBS representation, generate surface triangulation and close the triangulation of each component to form watertight polyhedra. *Interactive*

Initial Mesh (1%) – Determine extent of computational domain, and generate geometry adapted Cartesian starting mesh. *50% Interactive, 50% Automated*

Adaptive Euler Solution (40%) – Use of UNIX level control scripts cycle between running the Euler flow solver and flow field adaptation steps. Scripts handle job control, job submission and posting of intermediate results for review by engineer. Note that approximately 80% of this

| Process | Interactive | Automated |
|---|---|---|
| CAD | 100% | 0% |
| Surface Grid | 100% | 0% |
| Initial Mesh | 50% | 50% |
| Adaptive Euler Soltn. | 5% | 95% |
| Post Processing | 70% | 30% |

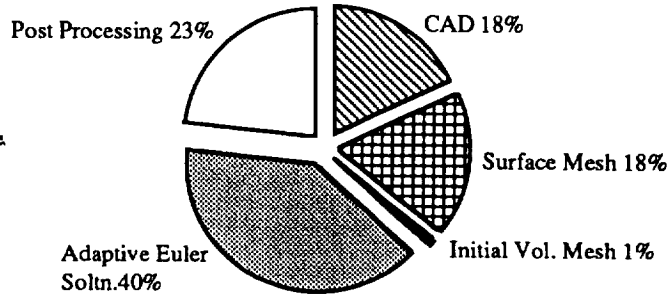



**Figure 8.** Breakdown of interactive and automated processes in HWT example in figure 9.



C-5.

time is dedicated to waiting for jobs in queue and not CPU. Also, this code is essentially unaccelerated and therefore converges slowly. *5% Interactive, 95% Automated.*

Post Processing (23%)– Post processing of results to extract surface pressure, cutting planes, streamlines in solution and other structures in flow. Surface pressures and cutting planes are extracted automatically. *70% Interactive, 30% Automated.*

An overall view of the discrete solution for this case is depicted by Figure 9. The computation placed 1.65M cells in the domain, and the adaptation was restricted at the finest level of adaptation to focus on the high-lift system. The figure shows a macroscopic view with isobars, streamribbons and an inset frame detailing the Cartesian mesh surrounding the flap system.

The total time for this simulation, in a research environment without dedicated CAD support or computing resources, was 17 days. This is relatively quick for a research application, but still far rom optimal for preliminary design or loads estimation environments. One strength is that subsequent parametric studies,(*e.g.* flap deflection or nacelle modifications) may be computed comparatively quickly. After approximately 1/2-1 day setup, follow on numerical simulations would turn around in a few days time.

II.C.3 Unresolved Issues and Current Research Topics

Viscous Simulations - Despite increasing documentation of efforts proposing full Navier-Stokes

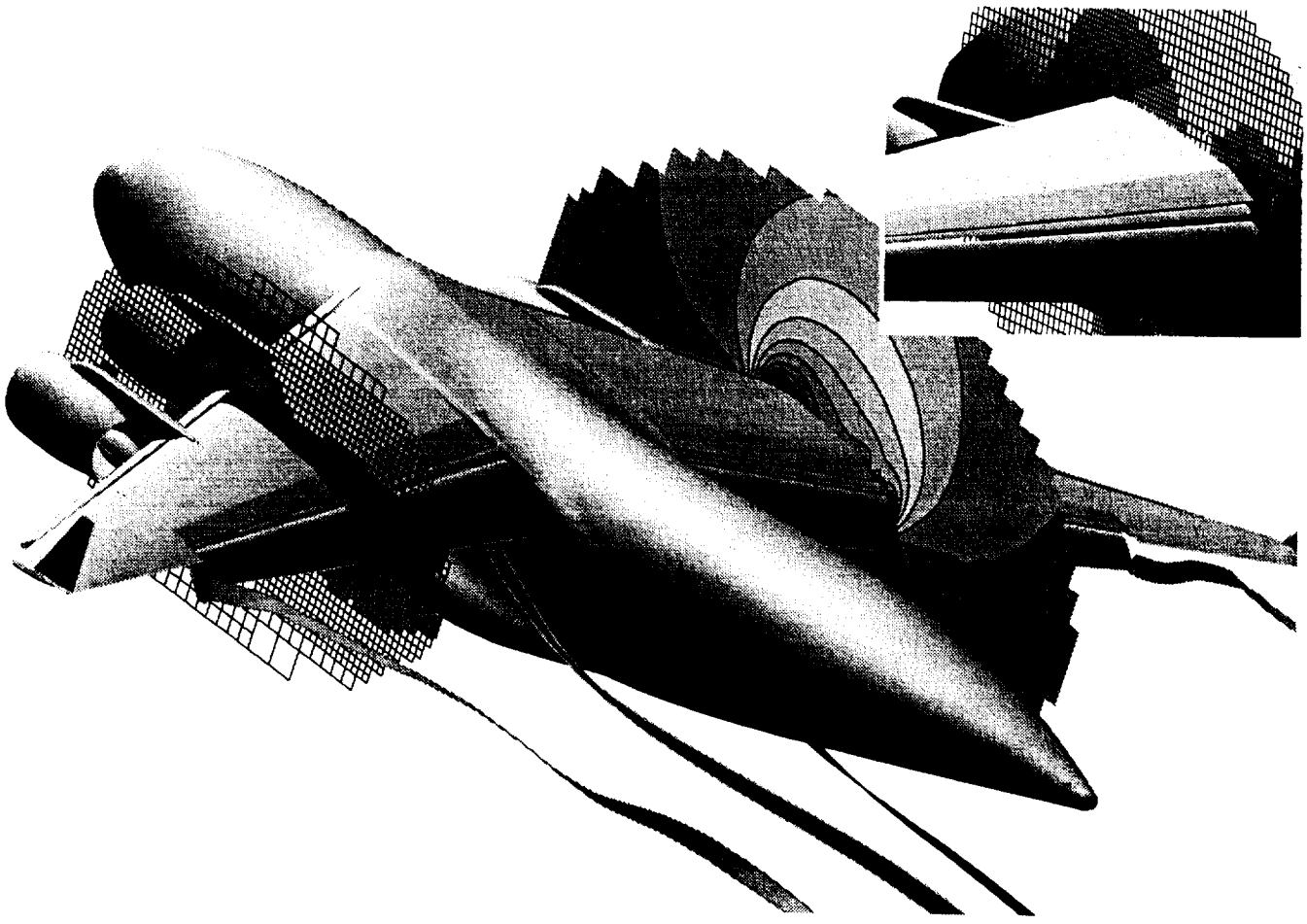


**Figure 9.** Isobars, mesh planes and streamribbons from High Wing Transport (HWT) case computed with flap, flap vane, spoiler, leading edge slat, winglet, pylons and nacelles. The final mesh contained 1.65M cells at 10 levels of refinement. The inset frame shows a detail of the computational mesh around the flap system.

simulations[68],[83],[84], open questions exist concerning efficiently constructed methods for viscous solutions on Cartesian meshes. Resolving boundary-layers with isotropic Cartesian cells is inherently inefficient. The use of prismatic grids near bodies inherits the difficulties associated with prismatic methods while adding the complexity of coupling the Cartesian and prismatic cells. As new techniques for constructing surface triangulations for entire configurations become more accessible, this path may prove fruitful. However, adapting Cartesian meshes to wakes and other viscous phenomena will continue to be a problem. Nearer term results may come through the coupling of Cartesian solvers with 3D boundary layer models.

Surface Modeling - Although surface modeling problems with Cartesian methods are somewhat alleviated by working directly from NURBS or triangulations of individual components, the example presented above shows that this procedure still demands considerable effort. Dedicated CAD systems depend on experienced users. Appropriate surface triangulations of various components are similarly non-trivial and require specialized software. Although no serious technical roadblocks exist in this process, the topic requires additional research attention and increased automation.

Accuracy at Boundaries - One view of Cartesian methods is that they take advantage of simpler flow field discretization stencils but suffer from increased boundary complexity. The intersection of the body with the grid may be very arbitrary thereby making the implementation of accurate and conservative boundary conditions more difficult. Although a variety of implementations have been reported, definitive formulations have been elusive. Most work has resulted in implementations that are between first and second-order accurate[71],[79],[81],[82].

## II.D. Automation of Block Structured Mesh Generation

Even with recent advancements in unstructured, hybrid, and Cartesian technology, the vast majority of CFD applications are computed with structured CFD schemes. This is especially true for Navier-Stokes simulations where the robustness, speed, and accuracy of unstructured viscous approaches have yet to be convincingly demonstrated. The basic paradigm of multi-block approaches begins with a decomposition of the computational domain in to a collection of deformed hexahedral blocks with either matching or overlapped faces. During the flow solution, the solver processes these blocks independently and then passes information to neighboring blocks as boundary conditions. Modifications of this basic model exist for both parallel machines and time dependent simulations.

In a recent paper, Dannenhoffer[85] separates the development of 3-D block structured grid generation into three stages of evolution. Initially, grid generation research focused on operations within each mesh block. Conformally mapped, elliptic and hyperbolic methods were developed for creating mesh block, were widely used[86],[87],[88]. Block-to-block pointers, common edges, and all other connectivity or control information was input by hand. Such systems typically operated in batch mode. With the fundamental algorithms identified, research focus then shifted toward useability. Second generation mesh generators replaced the tedious and error-prone process of typed-in control information with input through a Graphical User Interface (GUI) and such systems make up the bulk of currently available grid generators. Such interfaces are designed to give feedback to the user while aiding in visualizing the grid creation process. The end result, however, is essentially the same user-specified location and connectivity information required by the first generation systems. Third generation systems involve automating the blocking and mesh generation process to reduce the dependence on human input and decision making.

In the past decade, a number of approaches have been presented for automating the mesh generation process. These attempts may be generally categorized as either *interactive*, *knowledge-based*, or *algorithmic*[89], and this section presents a sampling of the current state of research in each of these directions.

II.D.1 Automation of Interactive Processes

The most direct approach toward automation of structured grid generation is through automation of the processes which exist in current gridding strategies. This implies that the user must still ultimately convey the topology of the body and block connectivity, and improvements stem from reducing the clutter of trivial tasks. Bookkeeping and obvious choices may be made automatically by the system, leaving the user free to work on the core problems of mesh and surface topology. One basic algorithm for interactive construction of blocked grids might be as follows:

**Algorithm I:**
a. Generate and link curves to form block edges.
b. Construct block faces from edges.
c. Distribute points on block edges and enumerate block connectivity.
d. Generate volume mesh within each block.

Some current systems take small steps toward automating this process by automatically dimensioning and distributing points on matching block faces, or by using geometric proximity to infer a topology and/or connectivity. However, even with such subtle streamlining, the process remains highly interactive and cumbersome. The essence of the problem is detail. As long as it requires an engineer to specify each curve of every block, the process will be labor intensive for complex configurations with many hundreds of blocks.

One solutions to this problem of excessive detail comes through the classical process of abstraction. Using a technique pioneered by Allwright[90] the user focuses on generating a blocked *abstraction* of the geometry which then communicates the topology of the configuration to the mesh generation system. Reference [85] has extended these ideas and applied them to a variety of published examples. One approach uses hypercube building blocks to construct "squared-up" abstractions of the configuration in three-dimensions. The use of the hypercube - which is simply the volume between two nested cubes - was first proposed by Allwright and applied to Euler simulations of a complete business jet as early as 1988[90]. Figure 10 shows a sample abstraction for the HWT configuration of Figures 6 and 7.

Once such an abstraction is formed, the blocking process occurs automatically following pre-set rules and point distribution algorithms. Next the surfaces of the abstraction may be deformed to match their counterparts in the real geometry. These deformations are distributed throughout the blocking, and thus the final block locations and topology are established. The last stage of mesh generation creates the volume grid within each block, using standard techniques.

The fundamental simplification in this approach is that the mesh topology is generated from an abstraction and not the actual geometry. This construction communicates only the *essential* geometric and topological information required to layout the mesh blocks. In addition, the process of forcing the user to construct the configuration out of pre-defined building blocks links certain surfaces with specific block layouts. In this way, the system guides the user, interactively, through the steps necessary to layout the mesh blocks. It is not clear if the procedure described will always lead to the most efficient blocking for any given configuration. However, the strength of the technique is that it will reliably lead to a decomposition. Ultimately this robustness and agility may provide shorter complete cycle times and yield a more efficient CFD process.

Variants of this approach were documented as early as 1988[90]. However, while prototype systems are in development at number of research centers, commercial products, detailed examples and extensive documentation are unavailable in the open literature.

II.D.2 Knowledge-Based Approaches

In the same paper which laid the groundwork for the geometric abstractions presented above, Allwright[90] investigated the utility of linking grid point positioning and spacing throughout the mesh to generic features of the configurations. Such concepts lead to so-called knowledge-based approaches and similar procedures have been investigated by a number of researchers and commercial firms. Applications of knowledge-based approaches revolve around the idea that a new configuration should be blocked only once and subsequent meshes for topologically similar configurations may then make use of similar block layouts and point distributions.

While conceptually simple, the ability to grid a new configuration from a library of previously studied examples is of indispensable utility in a detail design or loads estimation environment. For example, adjusting the position or contour of a pylon/nacelle combination, or investigating various flap deflections are common operations which should not require the re-blocking of entire mesh systems. The current software releases of many mesh generation software systems include this functionality. Often described as "macros", "configuration libraries" or "checkpointing" such approaches all permit the engineer to rapidly grid topologically similar cases with a minimum of new input.

The concept of "driving" mesh generation systems from an experience base has been extended by Buning[25] who has investigated prototype software which generates entire mesh systems, automatically runs flow simulations, and then extracts pre-determined results from the discrete solutions. These software "scripts" run at the operating system level and call FORTRAN or C codes which search for certain geometric or topological features in the input geometry. The system has been demonstrated for wing-body geometries using the OVERFLOW[91] solver and the Hypgen[92] mesh generation systems and is being

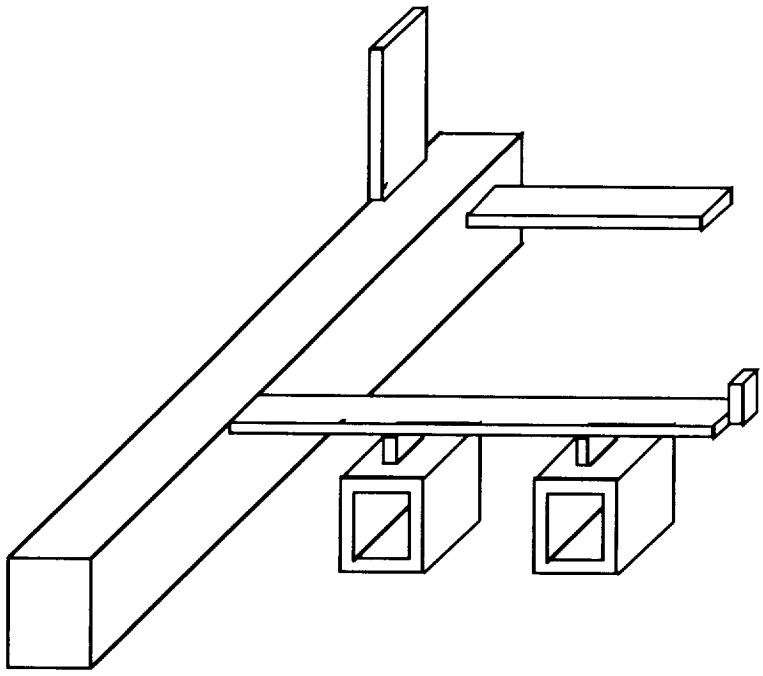


**Figure 10.** Sample blocked abstraction of HWT configuration following references [90] and [85].

considered for extension to more general classes of problems.

Extensions of these concepts have motivated research into controlling the blocking and point distribution process with expert systems or artificial intelligence. Preliminary work in this direction was performed by Dannenhoffer in 1991[93] and the topic was re-visited in a NASA sponsored workshop in late 1994[94]. Within such a framework, expert systems would either choose from libraries of pre-defined blocking strategies, or create new block layouts according to a pre-defined rule base.

II.D.3 Algorithmic Approaches

In the introduction of a paper which proposes a possible theoretical framework for formalizing the problem of multiple block structured grid generation, Cordova[89] poses the following question:

"Could it be that human intervention is necessary for solving the blocking problem?"

When one considers that unstructured mesh generators routinely operate with only minimal user input, the obvious answer to this question must be that such intervention can not be necessary. This argument is illuminating because it highlights the difference between our approach to multi-block gridding and techniques for unstructured mesh generation. Just as mesh blocking is a domain decomposition into hexahedra, unstructured mesh generation is a decomposition into tetrahedra.

Following this reasoning, Cordova suggests viewing structured grid blocking as a problem in Topological Graph Theory[95]. Within this framework, one seeks a hexahedral tessellation of the computational domain, in which each cell constitutes a grid block - or even a further decomposition to yield individual grid cells. Using the language of Computational Geometry (CG), the unstructured and structured mesh generation problems may be phrased as follows:[89]

Unstructured Mesh Generation - Decompose a region bounded by algebraic surfaces of degree = 1 into tetrahedra.

Block Structured Mesh Generation - Decompose a region bounded by algebraic surfaces of degree $\geq 1$ in to deformed hexahedra.

The inequality in the second statement recognizes that a block on a wall boundary will not necessarily have planar faces. We note in passing that unlike tetrahedra, hexahedra are not rigid figures (polyhedra). This property,, and the inequality in the second statement leads to the observation that while the CG problem of tetrahedral meshing is linear, the block structuring problem is non-linear, and solution strageties may require linearization steps.

With these definitions, Ref. [89] suggests the following algorithm for approaching multiple block grid generation:

**Algorithm II**:
a. Approximate the configuration with polyhedra.
b. Decompose surrounding space into convex polyhedra.
c. Construct computational space.
d. Set boundary conditions in each block
e. Solve elliptic equations globally over computational space.

Before examining this algorithm, we return to a brief analysis of the complexity of operations involved in the construction of tetrahedral grids as discussed in section II.A. The process of triangulating given coordinate data has linear complexity - that is to say that the operation count required to triangulate $N$ vertices scales linearly with the number of vertices. Sorting $N$ vertices requires $O(N\log N)$ operations and procedures such as clipping or the removal of hidden surfaces have quadratic complexity ($O(N^2)$). Such processes are in the class P since they may be performed in *polynomial time*. Other processes cannot be performed in polynomial time, and in fact their complexity is undetermined. These problems are said to belong to the class NP if candidate solutions may be verified in polynomial time[89][96]. A problem is

terrmed NP-complete when one can show that if it *could* be solved in polynomial time then all similar NP problems could also be solved with similar complexity.

In applying the decomposition step (b) of Algorithm II to 2-D blocking problems, one finds that decomposition of a multiply connected region (a polygon with polygonal holes in it) into quadrilateral regions is NP-complete[97]. It is interesting to note that if the region hapens to be simply connected (no holes) it may be quadrilateralized in polynomial time. However, NP-completeness does not imply that the problem is unsolvable, simply that one must search for a novel algorithm for generating candidate solutions. For example, a novel algorithm for quadrilateral decompositions in the plane may be obtained by first obtaining a candidate solution through a Delaunay triangulation of the coordinate data (possible in linear time). Then one simply inserts new sites at the centroids of each triangle, and connects them to new sites inserted at the midpoint of each edge. The result will always be a quadrilateral decomposition of the region, despite the problem's inherent difficulty due to its NP-completeness[98]. Unfortunately, decompositions constructed through this method are unlikely to meet the requirements of mesh smoothness that are imposed by the relatively low-order (linear) elements used by most CFD solvers.

In reference [99] Cordova suggests pursuing similar novel algorithms for solving the blocking problem. In addition the work in references [89] and [100] show that the framework and strategies of computational geometry and the theory of NP-completeness may be applied to the problem of point placement which is necessary to construct the computational space (step c, algorithm II) for a given mesh.

These examples demonstrate that while research in algorithmic approaches to solving the blocking and point placement problems is still formative, the work is promising and worthy of pursuit. Significant theoretical resources have been developed in the fields of Topological Graph Theory and Computational Geometry which are directly applicable to the problems of constructing structured multi-block meshes. Such approaches offer the possibility of replacing the heuristics of interactive mesh generation with robust, provable techniques.

## III. OPPORTUNITIES IN DESIGN AND APPLIED AERODYNAMICS

With the partial review of emerging CFD tools in the preceding section, focus now shifts to the application of these techniques within design and applied aerodynamics environments. Throughout this discussion a recurring theme will be the appropriateness of physical modeling and numerical tools as measured by the temporal requirements of the process. The examples and arguments in the introduction (sect. I), point out that various aspects of design and applied aerodynamics require a different balance of speed and accuracy. In reference [21] Rubbert uses an economic model of the aircraft industry to make a strong case that often a faster, but more approximate, process may lead to a higher level of functional "goodness" if that process is time constrained. In economic terms, he notes that maximum market share is generally achieved before a process has had time to reach its asymptotic level of functional goodness. Ultimately, the time asymptotic level of goodness achievable by a given process may mean very little if it takes to long to get close to that level. In the context of numerical simulations, the speed of a process is related to set-up time, CPU/memory requirements and post processing of discrete solutions.

As various CFD technologies mature, set-up and execution times will content to decrease. Thus, increasingly sophisticated physical modeling will become available progressively earlier in the design cycle. The discussion in this section is intended, therefore, to reflect the status of CFD research and computing hardware that currently exists.

### III.A Inviscid Techniques

Since they require minimal set-up, and avoid the problems of laying out and constructing a block structured mesh, unstructured and Cartesian approaches currently offer an attractive method for computing inviscid simulations. For example the unstructured, LWT simulation that was presented in figure 4 demonstrated that

a one million node Euler simulations currently require between one and two hours of supercomputer CPU. Alternatively, 200,000 vertex simulations currently converge in less than an hour on modern engineering workstations. The documented success of these methods suggests that 3D unstructured or Cartesian Euler solvers could be coupled with strip boundary layer modeling to provide an extremely flexible platform for rapid aerodynamic analysis. Such a system may be of greater immediate use than a full unstructured Navier-Stokes scheme, since inviscid solutions require approximately an order of magnitude fewer nodes, and 5-10 times less storage and CPU. Moreover, the discussion in section II points out that there are many unresolved issues remaining in the extension of these techniques to viscous simulations, while the inviscid technology is relatively mature.

Readily accessible inviscid, or inviscid and boundary-layer analysis is extremely valuable in various aspects of design and applications aerodynamics. The task of preliminary loads estimation is followed in series by that of structural analysis. This serial connection, and the importance of the follow-on task makes load estimation an example of a time critical process which would benefit greatly from improved physical modeling. Although it is true that many critical loading conditions occur in regions of the flight envelope in which Euler and boundary layer modeling may not be entirely sufficient, the approach offers great improvements over current approaches which are largely based on panel and potential codes. With the ability to capture non-linear features in the flow, Euler modeling permits prediction of vortex trajectories and shock structures which impact component loads over a wide range of conditions. With finer mesh resolution or solution adaptation, the same system would also provide a valuable method for getting preliminary results for detailed design or propulsion integration (PI) studies. With much of the set-up already completed during loads estimation, these fine mesh runs would guide PI teams in developing high quality block structured grid systems for full Navier-Stokes simulations. This preliminary analysis would alert design engineers to possible unfavorable shock or vortex interactions and permit *a priori* tailoring of the viscous mesh.

A critical aspect of such an approach is integration. Many manufacturers already use some kind of Euler-boundary layer modeling, but typically such systems are constrained by excessive overhead and are frequently poorly integrated. Automated mesh generation, macros for running angle of attack sweeps, automated post-processing, etc. are all features central to the usefulness of such a system. The engineer must be given quick access to the important data and the ability to assess the quality of the discrete solutions.

### III.B Surface Modeling and Preliminary Design

In surveying the techniques discussed in Section II, deficiencies in surface modeling emerged as an impediment to nearly all techniques for applied CFD. In this context, "surface modeling" refers to the process of generating an accurate representation of the true geometry in a form directly amenable to mesh generation. Unstructured, Cartesian and prismatic approaches require constrained triangulations of either the aircraft's surface or individual components. Block structured solvers require $i,j$ ordered surface meshes. Robust techniques for generating these descriptions have been documented in the literature, and are generally coupled with mesh generation software. However, transitioning this task to preliminary design or CAD software may offer significant advantages. Multi-disciplinary preliminary design and CAD systems contain a complete description of the geometry, and provide access to this information for all subsequent disciplines in the analysis cycle. Importing a CAD file into a mesh generation system places the burden of surface modeling not only on the mesh generator, but also on the CAD engineer. Since it is unlikely that the CFD model will include the myriad of details that the full CAD model includes, the model will need to be "cleaned-up" within the CAD system. Moreover, if a configuration originates in a preliminary design system, this model must first be translated to a dedicated CAD system before production of a model for grid generation. The intermediate step is unnecessary. With surface triangulations or surface grids directly available as output options form preliminary design or CAD software the current, updated geometry would be immediately available for aerodynamic analysis.

Preliminary design is an inherently multi-disciplinary process. A variety of software is in use, and many include multi-disciplinary optimization strategies that "fly" candidate vehicle designs through specified mission profiles, permitting multi-point optimization and trade-off analysis[101],[102],[103]. Physical modeling in these systems is based on an internal representation of the geometry, and includes approximate methods for structures, manufacturing and aerodynamics. The requirement for speed drives such systems to use handbook, or panel methods for approximate aerodynamic analysis. If such systems were extended to output constrained surface triangulations it would then be possible to also spawn coarse grid, Euler or full-potential CFD solutions using either unstructured or Cartesian meshes. Critical points within the flight envelope could be identified and run within the preliminary design environment, offering improved physical modeling. Preliminary design could then operate further from its experience base to explore novel configurations while simultaneously increasing the confidence in the results obtained with lower order methods.

### III.C Navier-Stokes Analysis

Processes such as wing optimization, propulsion integration, and high-lift system design ultimately require full Navier-Stokes analysis. Although the technology is constantly changing, block structured techniques currently appear to offer the most confidence and efficiency for such analysis. Many recent research programs have been dedicated to improving the efficiency of the grid generation process. The review of current and prospective block structuring techniques in Section II permits general statements concerning structured grid generation. Specifically, while developers strive for general algorithmic solutions to domain decomposition and point placement problems, interactive approaches have yielded remarkable success for specific topologies. Various research efforts have produced streamlined interactive procedures which guide even novice users through the block generation process using minimalist descriptions of complex aerospace geometries. Such approaches have demonstrated mesh set-up times on the order of hours using engineering workstations. Meanwhile, experience with these and more traditional mesh generation systems has resulted in a large knowledge base, or library, linking various mesh block arrangements to specific surface topologies. Work has also explored using this knowledge base to mesh new configurations by deforming topologically similar blockings from previous, or generic configurations. Clearly the possibility exists to combine these factors into grid generation systems which are vastly improved over those currently available.

If surface meshes were available directly from CAD products, or from prior steps in design, such mesh generation systems would offer greatly reduced set-up times. Even if appropriate surface meshes are not available, the use of abstractions discussed in Section II would permit block layout to be performed in parallel with surface mesh preparation - reducing at least the calendar time required for multi-block structured analysis.

Off-design analysis, wing optimization, and various tasks in applied CFD require the consideration of a variety of specific cases. This fact underscores the necessity of integrated CFD software to accept macroscopic automation. Automated control of job submittal, job monitoring and routine post-processing would again lead to more efficient processes.

## IV. CONCLUSIONS

This paper has reviewed the progress of a variety of emerging technologies against the issues which limit CFD's usefulness in design and applied aerodynamics. In doing so, the discussion has concentrated on approaches which intend to reduce the overhead associated with flow simulations around complex configurations. The discussion examined the current status of unstructured, hybrid, and Cartesian approaches as well as techniques for automating traditional multi-block mesh generation schemes. These methods were evaluated with special consideration of the differences that exist between research and design environments.

The rapidly maturing state of unstructured and Cartesian based techniques suggests the possibility of highly automated Euler-boundary layer solvers for use in loads estimation and other time-critical processes within the design cycle. Mesh generation for these techniques is largely automated and it requires only the generation of constrained triangulations for surface modeling as an input. Similarly, opportunities exist to increase the level of automation in the construction of blocked meshes for use with structured Navier-Stokes solvers. Strategies which link block structured mesh generation algorithms with libraries of prior examples are particularly attractive, since they avoid repeated meshing of topologically similar configurations. The interactive approach of building grid abstractions is also promising since it permits the blocking process to go on in parallel with surface modeling efforts. Work in algorithmic based auto-blocking suggests that domain decomposition and point placement operations in multi-block mesh generation may be properly posed as problems in Computational Geometry. This approach is unifying since it describes both multi-block structured methods and unstructured mesh methods within a common framework.

## V. REFERENCE LIST


1 Courant, R., Isaacson, E., and Reeves, M. "On the Solution of Nonlinear Hyperbolic Differential Equations by Finite Differences," *Comm. Pure and Applied Mathematics*, **5**:243-55, 1952.

2 Lax, P.D., and Friedrichs. "Weak Solutions of Non-Linear Hyperbolic Equation and Their Numerical Computation," *Comm. Pure and Applied Mathematics*, **7**:159-93, 1954.

3 MacCormack, R.W., "The Effect of Viscosity in Hypervelocity Impact Cratering," *AIAA Paper 69-354*, 1969.

4 Briley, W.R., McDonald, H. "Solution of the Three-Dimensional Navier-Stokes Equations by an Implicit Technique," *Proc. Fourth International Conference on Numerical Methods in Fluid Dynamics, Lecture Notes in Physics*, **35**, Berlin: Springer.

5 Beam, R.M., and Warming, R.F., "An Implicit Finite-Difference Algorithm for Hyperbolic Systems in Conservation Law Form," *J. of Computational Physics*, **22**:87-109. 1976.

6 Jameson, A., Schmidt, W., and Turkel, E. "Numerical Simulation of the Euler Equations by Finite Volume Methods Using Runge-Kutta Time Stepping Schemes," *AIAA Paper 81-1259*, AIAA 5th Computational Fluid Dynamics Conference. 1981.

7 Engquist, B., and Osher, S. "Stable and Entropy Satisfying Approximations for Transonic Flow Calculations," *Mathematics of Computation*, **34**:45-75. 1980.

8 Osher, S., "Riemann Solvers, the Entropy Condition, and Difference Approximations," *SIAM J. Numerical Analysis*, **21**:955-84, 1984.

9 Roe, P.L. "The Use of the Riemann Problem in Finite Difference Schemes," *Lecture Notes in Physics*, **141**:354-359, Berlin: Springer-Verlag, 1981.

10 Roe, P.L. "Approximate Riemann Solvers, Parameter Vectors and Difference Schemes," *J. Computational Physics*, **43**:357-72. 1981.

11 Steger, J.L., "Implicit Finite Difference Simulation of Flow About Arbitrary 2-D Geometry," *AIAA J.*, **17**:679-686, 1978.

12 Benek, J.A., Buning, P.G., and Steger, J.L., "A 3-D Chimera Grid Embedding Technique," *AIAA Paper 85-1523-CP*, AIAA 7th Computational Fluid Dynamics Conference, Cincinnati OH, July, 1985.

13 Augraud, F., and Dervieux, A., "Some Explicit Triangular Finite Element Schemes for the Euler Equations," Int. J. Numerical Methods in Fluids, **4**:749-64, 1984.

14 Jameson, A., Baker, T.J., and Weatherill, N.P., "Calculation of Inviscid Transonic Flow Over a Complete Aircraft," *AIAA Paper 86-0103*, 1986.

15 Shang, J.S., and Scherr, S.J., "Navier-Stokes Solution for a Complete Re-entry Configuration," *J. of Aircraft*, **23**(12):881-888, 1986. also *AIAA Paper 85-1509*, July, 1985.

16 Huband, G.W., Rizetta, D.P., and Shang, J.S., "Numerical Simulation of the Navier-Stokes Equations for an F-16A Configuration," *J. of Aircraft,* **26**(7):634-640, 1989. also *AIAA Paper 88-2507,* June, 1988.

17 Huband, G.W., Shang, J.S., and Aftosmis, M.J., "Numerical Simulation of an F-16A at Angle of Attack," J. of Aircraft, Vol. 27, **10**:886-892, 1990., also *AIAA Paper 90-0100,* Jan., 1990.

18 Flores, J., and Chaderjian, N.C., "A Zonal Navier-Stokes Methodology for Flow Simulation About a Complete Aircraft.," *AIAA Paper 88-2507,* Jun., 1988.

19 Cosner, R. "Issues in Aerospace Application of CFD Analysis," *AIAA Paper 94-0464,* Jan., 1994.

20 Garner, P.L., Meredith, P.T., and Stoner, R.C., "Areas for Future CFD Development as Illustrated by Transport Aircraft Applications," *AIAA Paper 91-1527-CP,* Jun., 1991.

21 Rubbert, P.E., "CFD and the Changing World of Airplane Design," *AIAA Wright Brothers Lecture,* Anaheim, CA, Sep. 1994.

22 Kallinderis, Y., Khawaja, A., and McMorris H., "Hybrid Prismatic/Tetrahedral Grid generation for complex Geometries," *AIAA Paper 95-0211,* Jan., 1995.

23 Nakahashi, K., "External Viscous Flow Computations Using Prismatic Grid," *Lecture Notes in Physics 414,* Thirteenth Int. Conf. on Num. Meth. in Fluid Dynam., Springer-Verlag, Berlin, pp. 280-284, 1993.

24 Pandya, S.A., and Hafez M.M., "A Semi-Implicit Finite volume scheme for Solution of Euler Equations on 3-D Prismatic Grids," *AIAA 93-3431,* Aug., 1993.

25 Buning, P.G., "CFD Process Automation Using Overset Grids," *NASA Workshop on Surface Modeling, Grid Generation, and Related Issues in CFD Solutions,* May, 1995.

26 Jameson, A., and Mavriplis, D.J., "Finite Volume solution of the Two-Dimensional Euler Equations on a Regular Triangular Mesh," *AIAA J.*, **24**:611-618, 1986.

27 Lerat, A., and Sides, J., "Numerical Simulation of Unsteady Transonic Flows Using the Euler Equations in Integral Form," *Israel J. Tech.,* **17**:302-310, 1979.

28 Venkatakrishnan, V., "Perspectives on Unstructured Grid Flow Solvers," *AIAA Paper 95-0667.* Jan., 1995.

29 Mavriplis, D.J., "Mesh Generation and Adaptivity for Complex Geometries," *Handbook o f Computational Fluid Mechanics,* Academic Press, Inc., San Diego, CA., 1995.

30 *Special Course on Unstructured Grid Methods for Advection Dominated Flows,* AGARD-R-787, May, 1992.

31 Desideri, J.A., and Dervieux, A., "Compressible Flow Solvers Using Unstructured Grids," *VKI Lecture Series, 1988-05*m pp.1-115, 1988.

32 van Leer, B., "Towards the Ultimate Conservative Difference Scheme V. A Second Order Sequel to Godunov's Method," *J. of Comp. Phys.*, **32**:01-136, 1979.

33 Durlofsky, L.J., Engquist, B., and Osher, S., "Triangle-Based Adaptive stencils for the Solution of Hyperbolic Conservation Laws," *J. of Comp. Phys.*, **98**:64-73, 1992.

34 Barth T.J., Jespersen, D.C., "The Design and Application of Upwind Schemes on Unstructured Meshes," *AIAA Paper 89-0366,* Jan., 1989.

35 Barth, T.J., "Numerical Aspects of Computing Viscous High Reynolds Number flows on Unstructured Meshes," *AIAA Paper 91-0721,* Jan. 1991.

36 Mavriplis, D.J., "Three Dimensional Multigrid for the Euler Equations," *AIAA J.*, **30**:1753-1761, 1992.

37 Luo, H., Baum J.D., Löhner, R., and Cabello, J., "Adaptive Edge-Based Finite Element Schemes for the Euler and Navier-Stokes Equations on Unstructured Grids," *AIAA Paper 93-0336,* Jan. 1993.

38 Löhner, R., Morgan, K., Peraire, and Vahdati, M., "Finite Element Flux-Corrected Transport (FCT) for the Euler and Navier-Stokes Equations," *International J. for Numerical Methods in Fluid Mechanics*, **7**, 1987.

39 Mavriplis, D.J., Jameson, A., and Martinelli, L., "Multigrid Solution of the Navier-Stokes Equations on Triangular Meshes," *ICASE Report 89-11, NASA CR 1817686*, Feb. 1989.

40 Frink, N.T., "Recent Progress toward a Three-Dimensional Unstructured Navier-Stokes Flow Solver," *AIAA Paper 94-0061*, Jan. 1994.

41 Delaunay, B., "Sur la Sphère Vide," *Bulletin of the Academy of Sciences of the USSR VII: Class. Sci. Mat. Nat.*, 7(6):793-800, 1934.

42 Peraire J., Vahdati, M., Morgan, K., and Zienkiewicz, O.C., "Adaptive Remeshing for Compressible Flow Computations," *J. of Computational Physics*, **72**:449-466, 1987.

43 Peraire J., Peiró, J. Formaggia, L., Morgan, K., and Zienkiewicz, O.C., "Finite Element Euler Computations in three Dimensions," *Int. J.. Num. Meth. in Engn.*, **26**, 1988.

44 Mavriplis, D.J., "An Advancing Front Delaunay Triangulation Algorithm Designed for Robustness," *AIAA Paper 93-0671*, Jan., 1993.

45 Rebay, S., "Efficient Unstructured Mesh Generation by Means of Delaunay Triangulation and Bowyer-Watson Algorithm," *J. Comp. Physics*, **106**(1):125-138, 1993.

46 Marcum, D.L., "Generation of Unstructured Grids for Viscous Flow Applications," *AIAA Paper 95-0212*, Jan., 1995.

47 Bern, M., and Epstein, D., "Mesh Generation and Optimal Triangulation," *Technic[illegible] CSL-92-1*, Xerox PARC, Mar., 1992.

48 Green, P., and Sibson, R., "Computing the Dirichlet Tessellation in the Plane," *The Computer J.*, **21**(2):168-173, 1977.

49 Müller, J.D., Roe, P.L., and Deconinck, H., "A Frontal Approach for Internal node Generation in Delaunay Triangulations," *Internat. J. of Num. Meth. in Fluids*, **17**, No. 3, pp.241-256, 1993.

50 Barth, T.J., "Steiner Triangulation for Isotropic and Stretched Elements," *AIAA Paper 95-0213*, Jan. 1995.

51 Barth, T.J., "Aspects of Unstructured Grids and Finite volume Solvers for the Euler and Navier-Stokes Equations," *Special Course on Unstructured Grid Methods for Advection Dominated Flows*, AGARD-R-787, pp.6.1-6.61, May, 1992.

52 Chew, L.P., "Guaranteed-Quality Triangular Meshes," *Technical Report TR 89-983*, Cornell Univ. Dept. of Comp. Sci., Mar., 1989.

53 Marcum, D.L., "Unstructured Grid Generation Using Iterative Point Insertion and Local Reconnection," *AIAA Paper 94-1926*, Jun., 1994.

54 Löhner, R., "Matching Semi-Structured and Unstructured Grids for Navier-Stokes Calculations," *AIAA Paper 93-3349*, Jun., 1993.

55 Löhner, R., "An Adaptive Finite Element Scheme for Transient Problems in CFD," *Comp. Meth. Appl. Mech. Eng.*, **61**:323-338, 1987.

56 Baum J.D., and Löhner, R., "Numerical simulation of Shock Interaction with a Modern Main Battlefield Tank," *AIAA Paper 91-1666*, Jun., 1991.

57 Baum, J.D., Luo, H., and Löhner, R., "Numerical Simulation of Blast in the World Trade Center," *AIAA Paper 95-0085*, Jan., 1995.

58 Venkatakrishnan, V., "Agglomeration Multigrid for the Three-Dimensional Euler Equations," *AIAA Paper 94-0069*, Jan., 1994.

59 Müller, J.D., "Quality Estimates and Stretched Meshes Based on Delaunay Triangulations," *AIAA J.*, **32**(12):2372-2379, Dec., 1994.

60 Aftosmis. M.J., Gaitonde, D.G., and Tavares, T.S., "On the Accuracy, Stability and Monotonicity of Various Reconstruction Algorithms for Unstructured Meshes," *AIAA Paper 94-0415*, Jan., 1994.

61 Barth, T.J., "Aspects of Unstructured Grids and Finite-Volume Solvers for the Euler and Navier-Stokes Equations," *von Karman Inst. for Fluid Dynam., Lect. Ser., 1994-05*, Mar., 1994.

62 Kallinderis, Y., and Ward, S., "Prismatic Grid Generation for 3-D Complex Geometries," *AIAA J.*, **31**(10):1850-1856, Oct., 1993.

63 Aftosmis, M.J., "Upwind Method for Simulation of Viscous Flow on Adaptively Refined Meshes," *AIAA J.*, **32**(2):268-277, 1994.

64 Parthasarathy, V., Kallinderis, Y., and Nakajima, K., "Hybrid Adaptation Method and Directional Viscous Multigrid with Prismatic-Tetrahedral Meshes," *AIAA Paper 95-0670*, Jan. 1995.

65 Quirk, J., "An Adaptive Grid Algorithm for Computational shock Hydrodynamics," PhD. Thesis, Cranfield Institute of Technology, 1991.

66 Melton, J.E., Enomoto, F.Y., and Berger, M.J., "3D Automatic Cartesian Grid Generation for Euler Flows," *AIAA Paper -93-3386-CP*, Jul., 1993.

67 Melton, J.E., Berger, M.J., Aftosmis, M.J., and Wong, M.D., "3D Applications of a Cartesian Grid Euler Method," *AIAA Paper 95-0853*, Jan., 1995.

68 Karman, S.L.Jr., "SPLITFLOW: A 3D Unstructured Cartesian/Prismatic Grid CFD Code for Complex Geometries," *AIAA 95-0343*, Jan., 1995.

69 Welterlen, T.J., and Karman, S.L.Jr., "Rapid Assessment of F-16 Store Trajectories Using Unstructured CFD," *AIAA 95-0354*, Jan., 1995.

70 De Zeeuw, D., and Powell, K., "An Adaptively-Refined Cartesian Mesh Solver for the Euler Equations," *AIAA Paper 91-1542*, 1991.

71 Berger, M., and Melton, J.E., "An Accuracy Test of a Cartesian Grid Method for Steady flow in Complex Geometries," Proc. *8thrIntl. Conf. Hyp. Problems*, Uppsala, Stonybrook, NY, Jun., 1995.

72 Purvis, J., and Burkhalter, J., "Prediction of Critical Mach Number for Store Configurations", *AIAA J.* **17(11)**, 1979.

73 D. Clarke, M. Salas, and H. Hassan, "Euler Calculations for Multi-Element Airfoils using Cartesian Grids," *AIAA J.l*, **24**, 1986.

74 Grossman, B., and Whitaker, D., "Supersonic Flow Computations using a Rectangular-Coordinate Finite-Volume Method," *AIAA Paper 86-0442*, Jan., 1986.

75 Gaffney, R., Hassan, H., and Salas, M., "Euler Calculations for Wings Using Cartesian Grids," *AIAA Paper 87-0356*, Jan.,1987.

76 Tidd, D. M., Strash, D. J., Epstein, B., Luntz, A., Nachson, A., and Rubin, T., "Application of an Efficient 3-D Multigrid Euler Method (MGAERO) to Complete Aircraft Configurations", *AIAA Paper 91-3236*, Jun., 1991.

77 Lednicer, D., Tidd, D., and Birch, N., "Analysis of a Close Coupled Nacelle Installation using a Panel Method (VSAERO) and a Multigrid Euler Method (MGAERO)," *ICAS-94-2.2.1*, 1994.

78 Levy, D., Wariner, D., and Nelson, E., "Validation of Computational Euler Solutions for a High Speed Business Jet", *AIAA 94-1843*, Jun., 1994.

79 Aftosmis, M.J., Melton, J.E., and Berger, M.J., "Adaptation and Surface Modeling for Cartesian Mesh Methods," AIAA Paper 95-1725-CP, Jun., 1995.

80 Berger, M., and LeVeque, R., "Cartesian Meshes and Adaptive Mesh Refinement for Hyperbolic Partial Differential Equations", *Proc. 3rd Intl. Conf. Hyp. Problems*, Uppsala, Sweden, 1990.

81 Berger, M., and LeVeque, R., "Stable Boundary Conditions for Cartesian Grid Calculations", *ICASE Report No. 90-37*, 1990.

82 Coirier, W. J., and Powell, K. G., "An Accuracy Assessment of Cartesian-Mesh Approaches for the Euler Equations", *AIAA Paper 93-3335-CP*, July, 1993.

83 Gooch, C.F., "Solution of the Navier-Stokes Equations on Locally-Refined Cartesian Meshes," PhD. Dissertation, Dept. of Aero. Astro. Stanford Univ., Dec., 1993.

84 Coirier, W.J., "An Adaptively-Refined, Cartesian, Cell-Based Scheme for the Euler Equations," *NASA TM-106754*, Oct., 1994. also Ph.D. Thesis, Univ. of Mich., Dept. of Aero. and Astro. Engr., 1994.

85 Dannenhoffer, J.F.III, "A new Method for Creating Grid Abstractions for Complex Configurations," *AIAA Paper 93-0428*, Jan., 1993.

86 Thompson, J.F., Warsi, Z.U.A., and Mastin, C.W., *Numerical Grid Generation, Foundations and Applications*, Elsevier Science Publishing Co., Inc., New York., 1985.

87 Sorenson, R.L., "The 3DGRAPE Book: Theory, User's Manual, Examples," *NASA TM-102224*, Jul., 1989.

88 Steger, J.L., and Rizk, U.M., "Generation of Three-dimensional Body-Fitted Coordinates Using Hyperbolic Partial Differential Equations," *NASA TM-86753*, 1985.

89 Cordova, J.Q., "Towards a Theory of Automated Elliptic Mesh Generation," NASA *Workshop on Software Sys. for Surf. Modeling and Grid Gen.*, Langley VA, Apr., 1992.

90 Allwright, S.E., "Techniques in Multiblock Domain Decomposition and Surface Grid Generation," *Numerical Grid Generation in Computational Fluid Mechanics '88*, Pineridge Press, 1988.

91 Buning, P.G., Chan, W.M., Renze, K.J., Sondak, D.L., Chiu, I.T., and Slotnick, J.P., "OVERFLOW User's Manual, Version 1.6ap, 23 March 1994," NASA Ames Research Center, Moffett Field, CA, Mar., 1994.

92 Chan, W.M., Chiu, I.-T., and Buning, P.G., "User's Manual for the HYPGEN Hyperbolic Grid Generator and HGUI Graphical User Interface," *NASA TM 108791*, Oct., 1993.

93 Dannenhoffer, J.F.III., "Computer-Aided block Structuring Through the Use of Optimization and Expert System Techniques," *AIAA Paper 94-1565*, Jun., 1991.

94 *Improving the Computational Fluid Dynamics Design Process*, Conf. Proceedings, NASA Ames Research Center., Nov., 8-9, 1994.

95 Gross, J.L., and Tucker, J.L., *Topological Graph Theory*, Wiley Interscience, 1987.

96 Garey, M.R., and Johnson, D.S., *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W.H. Freeman and Co., 1979.

97 Keil, M.J., and Sack, J.R., "Minimum Decomposition of Polygonal Objects," *Computational Geometry*, G. Toussaint, Ed., Elsevier Science, 1985.

98 Lubiw, A., "Decomposing Polygonal Regions Into Convex Quadrilaterals," *Proc. 1st Symp. on Comp. Geom.*, ACM, 1985.

99 Cordova, J.Q., "Computational Geometric Aspects of Automated Mesh Generation," *Int. J. Comp. Geom. and App.*, 1992.

100 Nobel, S.S., and Cordova, J.Q., "Blocking Algorithms for Structured Mesh Generation," *AIAA Paper 92-0659*, Jan., 1992.

101 Gregory, T.J., "Computerized Preliminary Design at the Early Stages of Vehicle Defination," *AGARD CP-147-***1**(147):6.1-6.8.,

102 Myklebust, A., and Gelhausen, P., "Improving Aircraft Conceptual Design Tools - New Enhancements to ACYSNT," *AIAA Paper 93-3970*, Aug., 1993.

103 Rivera, F. Jr., and Jayaram, S., "An Object Oriented Method for the Defination of Mission Profiles for Aircraft Design," *AIAA Paper 94-0867*, Jan. 1994.

# AN ASSESSMENT OF UNSTRUCTURED GRID TECHNOLOGY FOR TIMELY CFD ANALYSIS

Tom A. Kinard
Lockheed Aeronautical Systems Company
Marietta, Georgia
and
Deanne M. Schabowski
Lockheed Fort Worth Company
Fort Worth, Texas

## SUMMARY

An assessment of two unstructured methods is presented in this paper. A tetrahedral unstructured method USM3D, developed at NASA Langley Research Center is compared to a Cartesian unstructured method, SPLITFLOW, developed at Lockheed Fort Worth Company. USM3D is an upwind finite volume solver that accepts grids generated primarily from the Vgrid grid generator. SPLITFLOW combines an unstructured grid generator with an implicit flow solver in one package. Both methods are exercised on three test cases, a wing, and a wing body, and a fully expanded nozzle. The results for the first two runs are included here and compared to the structured grid method TEAM and to available test data. On each test case, the set up procedure are described, including any difficulties that were encountered. Detailed descriptions of the solvers are not included in this paper.

## INTRODUCTION

One of the aims of computational fluid dynamics (CFD) is the timely analysis of complete aircraft configurations. To this end, unstructured methods hold considerable promise as a tool by which CFD engineers can efficiently analyze complete aircraft configurations in a timely fashion. Although structured grid methods such as TEAM[1] and CDFALCON[2], based on patched multi-block grids have been applied to complete configurations like the F-22 and F-16, the time to generate such grids remains unacceptably large. In order to reduce the time required to generate grids around complex configurations, unstructured grid technology are being explored. The goal is to reduce turnaround time from weeks to days to hours.

Two unstructured methods are currently being used at Lockheed. The first method, acquired from NASA Langley, is composed of three codes, a grid preprocessor GridTool[3], an advancing front grid generator Vgrid[4], and an Euler flow solver USM3D[5]. The second unstructured method, SPLITFLOW[6], is being developed at the Lockheed Fort Worth Company (LFWC) and uses Cartesian unstructured meshes. This paper compares and contrasts these unstructured methods based on three test cases. The results are compared to available experimental data and to results generated by the patched structured grid method TEAM.

Copyright ©1995 by Lockheed Corporation. All rights reserved.

## SYMBOLS

| | |
|---|---|
| C | Chord |
| $C_D$ | Drag coefficient |
| $C_L$ | Lift coefficient |
| $C_M$ | Pitching moment coefficient |
| $C_N$ | Normal force coefficient |
| CFL | Courant Friedrichs Lewy number |
| M | Mach number |
| x,y,z | Cartesian coordinates |
| $\alpha$ | Angle of attack |

## METHOD DESCRIPTION

### Tetrahedral Unstructured Method

The tetrahedral grid generation system from NASA Langley is composed of three codes. With these codes, an Euler solution can be generated on simple configurations in a matter of hours.

GridTool. The first code used in the unstructured grid generation process is GridTool. This program takes geometry files in either discrete point or IGES[7] format. Once an adequate geometry file is entered into the program, a user interactively constructs curves and patches on the surfaces exposed to the flowfield. When the entire surface has been divided into patches then the outer boundaries are prescribed, usually with a simple box. Point and line sources are then prescribed, which control the distribution of points not only on the surface but also in the flowfield. A restart option is available to allow the engineer to save intermediate results. This option is particularly helpful in treating complex configurations which may require more than one session to complete the patching. The output of GridTool is an input file for Vgrid. GridTool also has the capability of displaying surface grids on a patch by patch basis, to allow the user to inspect the quality of an unstructured mesh.

Vgrid. The surface and volume grids are generated with Vgrid. Vgrid uses the advancing front method[4] to generate both surface mesh and volume meshes. A structured background[8] mesh is used to define the point distributions for the surface and volume region. The structured background grid is constructed by subdividing the entire flowfield domain into cells. The spacing information for the unstructured grid is stored at the cell nodes. The distributions are determined in a manner similar to the diffusion of heat in a conducting medium from discrete sources.

Once the background grid has been created a surface mesh is constructed by placing points along the edges of the user defined patches, and then triangles are constructed to fill each patch. After each patch is triangulated, the mesh quality is checked automatically and any regions of poor quality are displayed. The user has the ability to change the patch in order to achieve a better meshing if necessary. The surface mesh then forms the initial front for the volume grid. The front is advanced into the field by introducing new points and forming tetrahedra and new faces to complete the grid. This step is usually accomplished in a batch process. The code continues to fill the flowfield domain until either the domain is filled with cells or no more cells can be formed thus leaving pockets or voids in the grid. These pockets are usually filled by removing a layer of cells around the pocket creating a larger void and a new front. The grid generator is restarted and cells are added until the grid is completed. In some cases the background grid has to be modified in order to achieve a complete grid. A grid quality check is then initiated and negative and skewed cells are reported and then corrected by removing cells around the bad cell and refilling. As with the incomplete grid, sometimes the background grid has to be modified in order to remove bad cells.

USM3D. Once an acceptable grid has been generated, the next step is to compute the flow solution using the Euler solver, USM3D. The solver, developed at NASA Langley and solves the time-dependent Euler equations for an ideal gas using a cell-centered finite volume formulation. Spatial discretization is accomplished by the use of Roe's flux difference splitting. The solutions are advanced in time by either an explicit multi-stage Runge-Kutta scheme or an implicit Gauss-Seidel scheme. Local time stepping is used to accelerate the convergence of the solution to a steady state by using a CFL number near the local stability limit. The maximum time step for the explicit scheme is enlarged by the use of implicit residual smoothing. USM3D supports boundary conditions commonly available to Euler solvers. The code is usually run on a Cray-type machine, but can easily be run on other high-end workstations with sufficient memory and computing speed. USM3D uses 44 words per cell of core memory and 26 $\mu$ sec per cell per cycle for the explicit version of the code and 180 words per cell of core memory and 64 $\mu$ sec per cell per cycle for the implicit version of the code. All computer times are for a Cray YMP.

## Cartesian Unstructured Method

SPLITFLOW is an unstructured Cartesian code developed by LFWC for analyzing complex 3-D geometries. SPLITFLOW generates cube-shaped cells that are aligned with the Cartesian coordinate axes. Boundary geometry is defined by triangular faces, or facets. At boundaries, cells are "cut" to account for volume and flux changes due to parts of the cells being inside of the solid surface. This feature allows SPLITFLOW to handle extremely complex geometries, and little care need be taken by the user to prepare or maintain the grid. This type of grid was used on all of the geometries presented in this paper.

Initial grid cell sizes are scaled from geometry facet sizes and are then refined or coarsened, at specified iteration intervals, by the solver based on the user's choice of gradient adaption functions (Mach number, pressure, etc.). The coarsening process uses statistical methods to look for low gradient regions in the flowfield from which to remove cells, thus reducing grid density and computational requirements. The coarsening process is limited by a grid smoothing algorithm which requires adjacent cells to be no more than one "generation" apart (Figure 1). Further, cells are deleted by groups of eight and only if all of the child cells in that group are flagged for coarsening. This is done to maintain the data structure. The refinement process follows, also applying statistical methods, and searches for high gradients to determine where cells need to be added. Grid refinement involves recursively sub-dividing each cell into eight cells which become "children" to the initial cell. Since the code is "smart" enough to place cells where they are needed, the best initial grid is usually sparse and the flowfield is used to determine where new cells should be placed. With a sparse initial grid, flowfield information can propagate in fewer iterations, each of which take less time because there are fewer cells. For example, a grid which is to be limited to 800,000 cells may be appropriately initialized to 50,000 -100,000 cells.

Another benefit of cutting boundary cells is that geometry changes can be made easily while salvaging a developed solution. For example, if the user has a converged solution of an aircraft with undeflected control surfaces, a new geometry model with deflected control surfaces can simply be substituted. SPLITFLOW will recut, or "mark", the appropriate boundary cells and continue solving and refining on the new geometry/flowfield; the cost-effectiveness of such a feature is clear.

SPLITFLOW includes a point implicit solver which typically brings about convergence in under 500 iterations. The amount of memory that is required to run the solver portion of SPLITFLOW is approximately 180 words per cell. The algorithm also includes automatic time step scaling based on the convergence of the sub iterations of the point implicit scheme. The definition has a direct influence on attainable time step size.

## WING C TEST CASE

Wing C is a low aspect ratio fighter-type wing. A geometric description of this wing is shown in Figure 2. Wing C was designed to have a large leading edge sweep and mean aerodynamic chord. At the design condition of 0.85 Mach and five degrees angle of attack , the wing has moderate aft loading, mild shocks and mild pressure recovery. The objective of this test case was to evaluate how well each method can model transonic flows with shocks. Extensive force and pressure data has been generated for this geometry on a large scale[9] model and a small scale model[10].

### Tetrahedral Unstructured Method Case Analysis

The surface geometry for this test case was generated from tabular airfoil sections and wing characteristic data. A discrete point data file was constructed and used as input to GridTool. Since Wing C has a round leading edge and tip, more patches were needed around these regions to help achieve good resolution. A total of 14 patches were used on the wing surface and outer boundaries. Two line sources were placed at the leading edge and slightly aft of the leading edge in order to achieve adequate resolution in this region. A line source was also placed along the trailing edge and along the chord line at the root and tip of the wing. Eight point sources were placed at the corners of the bounding box to control the spacing in the far fields. The resulting grid contained 47,390 nodes and 266,101 cells. A plot of the upper surface and symmetry plane grid is shown in Figure 3. The cell volumes ranged from a minimum of .1226E-08 near the tip leading edge to .1026E+01 at the outer boundary. The time needed to generate this grid from the initial input geometry was four hours.

With the grid completed, a flow solution was obtained using USM3D at a Mach number of 0.85 and an angle of attack of five degrees. Three boundary conditions were used in this Wing C analysis, a far field, symmetry, and solid boundary condition. The explicit version of the code was used in this test case. The CFL number used for this run was set to four with a smoothing coefficient set to one half. The normal execution procedure for USM3D is to let the code pick when to use low and higher order flux difference splitting. For this case 287 cycles were executed on lower order differencing, or until one order of convergence was achieved. The solution then ran in higher order differencing until 2000 total cycles were reached. At the end of 2000 cycles the residual had reduced by two and a half orders. A plot of the residual convergence is shown in Figure 4. USM3D requires 44 words per cell of memory regardless of what platform chosen to run the code. However, the code performance depends on the platform chosen to run the code. On a Cray YMP the code executes at 26μ seconds per cell per cycle. On a HP 755 workstation the code runs at around 290μ seconds per cell per cycle. For this case, the total solution time was 1.7 Cray C90 hours or 11μ sec per cell per cycle.

### Cartesian Unstructured Method Case Analysis

The geometry for the Wing C test case was developed from the surface grid used in the above analysis. The quadrilateral cell database was converted to a faceted surface that was compatible with SPLITFLOW. This conversion process from discrete data to a faceted geometry file required around a half an hour. The facet model contained 10,970 facets. The geometry was a half-wing model with a plane of symmetry. After the faceted geometry has been generated, the time required to start SPLITFLOW is around twenty minutes. The initial grid consisted of 225,281 cells on 18 grid levels. The initial grid size was large due to the extremely curved trailing edge. A plot of the initial grid at the symmetry plane and the upper surface of the wing is shown in Figure 5. The faceted model required a half hour to generate and the input and job file for SPLITFLOW required an additional twenty minutes to prepare.

Starting with this grid, a flow solution was obtained using SPLITFLOW at a Mach number of 0.85 and

an angle of attack of five degrees. Three boundary conditions were used in this Wing C analysis: a far field, symmetry and tangent flow for the surface boundary condition. SPLITFLOW has automatic CFL adjustment. In this case the CFL number was limited to 30. The grid adaption occurred every 50 iterations with a final grid consisting of 445,521 cells on 20 grid levels. SPLITFLOW required 4.8 CPU hours on a Cray C90. Grid generation required 0.32 hours out of that total.

Adaptive refinement of the grid was based on gradients of Mach number and pressure. A cutting plane through the 70 percent span location shown in Figure 6 reveals the resolution of the grid near the multiple shocks on the upper surface of the wing.

### Solution Comparisons

Figure 7 show comparisons of surface pressures from USM3D, SPLITFLOW, TEAM, and test data. The pressures predicted by TEAM, USM3D, and SPLITFLOW are almost identical except for minor differences near shocks. Both unstructured solutions compare well with the test data along the lower surface and ahead of the shocks. Aft of the shocks there is shock induced separation which Euler codes cannot model. The lack of adequate resolution near the stagnation point is the most likely cause of USM3D not being able to match the stagnation pressure shown in the test data. At the 70 percent span station, the solution from USM3D shows the most difference when compared to TEAM and SPLITFLOW. This is due to inadequate grid resolution at this station. As one moves closer to the tip, the grid used in USM3D becomes finer and yields a better solution, as seen at the 90 percent station. Table I contains a comparison of the computed forces and moments and experimental data. USM3D shows good agreement with TEAM on drag and pitching moment and only slight difference on lift and normal force. USM3D compared well the test normal force. SPLITFLOW predicts a higher drag than either USM3D or TEAM, but is in good agreement on the normal force.

## ARROW WING BODY TEST CASE

The second geometry used in this study was an arrow wing body configuration[11]. A schematic of this geometry is shown in Figure 8. The wing on this configuration has a round leading edge and was designed for efficient supersonic cruise. It has both leading and trailing edge flaps. The tip of the arrow wing body configuration is closed by a flat plate. The body has a circular cross-section with a straight centerline. A deflection of 8.3° was imposed on the entire trailing edge flap. The leading edge flaps were kept in the faired or zero deflection configuration. The flow conditions for this case were chosen to be a Mach number of 0.85 and an angle of attack of four degrees. At these conditions the wing is essentially shock free.

### Tetrahedral Unstructured Method Case Analysis

The starting geometry for this case was taken from an existing TEAM structured grid. The surface mesh, which consists of discrete point data, was used as input to GridTool. A total of 115 patches were required to completely cover the configuration, sting and far field boundaries. The outer boundaries were located 12 spans from the body. Particular attention was paid to the leading edge, wing-fuselage and wing-flap intersections. Even though care was taken to preserve the rounded character of the leading edge, toward the tip the leading edge of the input geometry becomes relatively sharp, the effect of which is reflected in the unstructured grid solution. Sources were placed along the centerline of the body, wing leading and trailing edges, and along the tip. An additional source was placed just aft of the leading edge in order to help add points in that region. Some difficulties were experienced near the tip where the cell sizes become extremely small compared to the characteristic length of the wing. Although no negative cells were found, there existed some highly skewed cells near the leading edge in the tip region of the grid. The final grid consisted of 57,788 nodes

and 307,677 cells. A plot of the surface grid is shown in Figure 9. The cell volumes ranged from 0.246E+07 near the outer boundary to 0.288E-06 near the wing tip leading edge. The set up time for this grid was 24 man hours.

As with Wing C, only symmetry, far and solid boundary conditions were used in this solution. The numerical parameters used in this run were a CFL of 4 and smoothing coefficient of a half. A three stage explicit scheme was used throughout this run. The code switched from lower order differencing to higher order differencing after 181 cycles. The first attempt to run this grid failed shortly after switching to higher order differencing. The pressure and density approached zero in the small cells that were around the wing tip leading edge. A modification to the code which set these cells to first order allowed the code to continue to run. A total of 461 cells were set to first order after 2000 cycles had been executed. After 2000 cycles the residual had reduced by two and a quarter orders. A plot of the residual convergence is shown in Figure 10. The total solution time for this case was 2.5 Cray C-90 hours. or 10μ sec per cell per cycle.

## Cartesian Unstructured Method Case Analysis

The geometry for the Arrow Wing Body test case was developed from a baseline IGES file. The model was modified to define a full span 8.3 degree trailing edge deflection. The model contained 34,034 facets defining one half of the aircraft. The initial grid consisted of 27,737 cells.

A SPLITFLOW flow solution was obtained at a Mach number of 0.85 and four degrees angle of attack. Three boundary conditions were used in this Arrow Wing Body analysis: a far field, symmetry and tangent flow for the surface boundary condition. The grid adaption occurred every 40 iterations with a final grid consisting of 169,749 cells. A pressure coefficient contour plot of the symmetry plane grid and surface geometry is shown in Figure 11. In this case, the CFL was limited to 8.0. SPLITFLOW required 0.69 CPU hours on a Cray C90. Grid generation required 0.11 of that total.

Although the solution was run to 197 iterations, the data shown in Figure 12 shows that the force and moment data were converged within engineering accuracy (assuming a data uncertainty band of +/- 0.05) at approximately 100 iterations. The grid at that point consisted of 60,000 cells. The CPU time at 100 iterations was 0.24 Cray C90 hours.

## Solution Comparisons

Figure 13 show comparisons of surface pressure coefficients from USM3D, TEAM, SPLITFLOW and test data. The effect of the flap defection can be seen at the inboard span station. The leading edge peak suction pressure from USM3D agrees better with the test data than the TEAM results, due to insufficient grid resolution in the TEAM grid. USM3D and SPLITFLOW agree well except at the 80 percent span station. The USM3D pressures generally agree with the test data except aft of the flap deflection. The SPLITFLOW pressures show similar behavior to the TEAM results in the region aft of the flap break. At the tip the results are less encouraging. In this region there is a small vortex that is not picked up by any of the codes due to the inviscid nature of the computations. The peak suction predicted by USM3D is much higher than TEAM or SPLITFLOW. Both TEAM and USM3D agree with the test data on the lower surface. On the upper surface aft of the first 10 percent, both USM3D and TEAM show similar behavior. The SPLITFLOW results show good agreement at the leading edge but show less agreement with test data or TEAM results especially near the trailing edge. Table II shows a comparison of computed forces and moments for all codes and test data. Only normal force and pitching moment numbers were available in the wind tunnel report. The normal force predicted by USM3D is higher than the test data and the normal force computed by TEAM. SPLITFLOW shows good agreement with TEAM on lift and normal coefficient. SPLITFLOW predicts a higher drag and lower pitching moment than either USM3D or TEAM.

## NASA 2D-CD NOZZLE TEST CASE

The objective of this test case was to assess each code's ability to correctly model flows internal and external to a 2D-CD[12] nozzle. A schematic of the nozzle chosen is shown in Figure 14. The nose-forebody section of the model followed a smooth external transition from a circular cross section at the conical nose to a superelliptical cross section at fuselage 26.5. The maximum external cross-sectional area of 41.17 $in^2$ occurs at fuselage station 26.5. The cross-sectional area and the external geometry remained constant from fuselage station 26.5 to fuselage station 55.05. For this study only the section from fuselage 26.5 to the exit of the geometry was modeled. The nozzle geometry has straight, parallel internal sidewalls. The nozzle-to-throat area ratio of 1.25 yielded a design exit Mach number of 1.6 and nozzle pressure ratio (ratio of the local total pressure to the free stream static pressure) of 4.25. Extensive tests were done on this model at NASA, but only one case will be examined here. An attached external flow test case was chosen for this study. The free stream Mach number was 0.6 zero degrees angle of attack, and a nozzle pressure ratio of 4.0.

At the time this paper was written, (February 1995) results from both SPLITFLOW and USM3D were not available. Grids had been generated, but flow solutions were proving to be a challenging undertaking. Both USM3D and SPLITFLOW are still under development and appropriate boundary conditions for modeling propulsion type flows are being investigated.

## CONCLUDING REMARKS

The use of unstructured methods can substantially reduce the time required to generate Euler solutions on complex configurations as compared to structured grid methods. Two unstructured methods were applied to three test cases to assess their strengths and weaknesses. The tetrahedral method USM3D showed good agreement and robustness on the two external flow test cases. SPLITFLOW, the Cartesian unstructured method, converged in fewer cycles than USM3D and showed the benefits of adaptive grid refinement.

The set up time for the two codes showed the most difference. SPLITFLOW required the least amount of time for problem set up. This is due to the fact that the grid generator is an integral part of the solver. Only the surface mesh needs to be input to the code. If IGES geometry is available to the engineer constructing the SPLITFLOW input files, the time to set up a problem is on the order of a half an hour. If discrete point data is used then the set up time can run into a number of hours. This is in contrast to USM3D which requires the full volume grid to be input to the solver. Depending on the complexity of the geometry this grid generation time can run from a few hours to a few days. For wing alone cases, the grid generation time is on the order of hours. For full configuration aircraft, the grid generation time can be on the order of a few days.

When the solvers are examined, comparisons are harder to arrive at. The memory needed to run SPLITFLOW is on the order of 180 words per cell. This can vary depending on the way the code is initialized. For USM3D, the memory is easier to determine since the grid generator is a separate code. Using the explicit option in USM3D, the memory required for execution is 44 words per cell, using the implicit option the memory increases to 180 words per cell. Since grid generation time is an integral part of SPLITFLOW the execution time in terms of time per cell per cycle is difficult to quantify.

A converged solution is achieved in USM3D when two orders of residual reduction and the forces and moments have stopped oscillating. A similar situation exists for SPLITFLOW, but the primary convergence criteria is forces and moments. Solutions are stopped when the forces and moments have reached a steady state.

On a wing alone test case both unstructured methods showed similar results. Both methods were shown to have the ability to model external flowfields with good accuracy. Further development of the solvers is required to treat the nozzle test case representative of a combined external / internal flow-field.

## REFERENCES


1. Goble, B.D., Raj, P., and Kinard, T.A., "Three-Dimensional Euler/Navier-Stokes Aerodynamic Method (TEAM), Version 713 User's Manual," WL-TR-93-3115, February 1994.

2. Reed, C.L., "Central Difference Falcon User's Manual," June 1994.

3. Abolhassani, J.S., "Unstructured Grids on Nurbs Surfaces", AIAA-93-3454, 1993.

4. Parikh, P., Pirzadeh, S., and Löhner, R., "A Package for 3-D Unstructured Grid Generation, Finite-Element Flow Solution, and Flow Field visualization," NASA CR-182090, September 1990.

5. Frink, N.T., Parikh, P., and Pirzedeh, S., "A Fast Upwind Solver for the Euler Equations on 3-D Unstructured Meshes," AIAA-91-0102, 1991.

6. Karman, S. L., "SPLITFLOW: A 3d Unstructured Cartesian/Prismatic Grid CFD Code For Complex Geometries", AIAA-95-0343, January 1995

7. "The Initial Graphics Exchange Specification (IGES) Version 5.0," Distributed by National Computer Association, Administrator, IGES/PDES Organization, 2722 Merrilee Drive, Suite 200, Fairfax, VA 22031.

8. Pirzadeh, S., "Structured Background Grids for Generation of Unstructured Grids by Advancing Front Method," AIAA-91-3233, 1991.

9. Keener, E.R., "Pressure-Distribution Measurement on a Transonic Low-Aspect Ratio Wing," NASA TM 86683, September 1985.

10. Hinson, B.L., and Burdges, K.P., "Acquisition and Application of Transonic Wing and Far-Field Test Data for Three Dimensional Computational Method Evaluation," AFSOR-TR-80-0421, March, 1980.

11. Manro, M.E., Manning, K.J.R., Hallstaff, T.H. and Rogers, J.T., "Transonic Pressure Measurements and Comparison of Theory to Experiment for Arrow-Wing Configuration", Volume II: Experimental Data Report-Effects of Control Surface Deflection,NASA CR-132728, October 1975

12. Compton III, W.B., Thomas, J.L., Abeyounis, W.K., and Mason, M.L., "Transonic Navier-Stokes Solutions of Three-Dimensional Afterbody Flows," NASA TM 4111, July, 1989.

Table I. Forces and Moments for Wing C

| Wing C<br>Mach = 0.85 $\alpha$ = 5.0° | | | | |
|---|---|---|---|---|
| | CL | CD | CN | CM |
| USM3D | 0.5407 | 0.03854 | 0.5420 | -0.07792 |
| SPLITFLOW | 0.5374 | 0.04026 | 0.5389 | -0.06813 |
| TEAM | 0.5650 | 0.03995 | 0.5653 | -0.07782 |
| TEST | | | 0.540 | |

Table II. Forces and Moments for Arrow Wing Body

| Arrow Wing Body<br>Mach = 0.85 $\alpha$ = 4.0° | | | | |
|---|---|---|---|---|
| | CL | CD | CN | CM |
| USM3D | 0.2748 | 0.01747 | 0.2812 | -0.1253 |
| SPLITFLOW | 0.2610 | 0.02484 | 0.2683 | -0.0857 |
| TEAM | 0.2666 | 0.02071 | 0.2674 | -0.0947 |
| TEST | | | 0.276 | -0.106 |

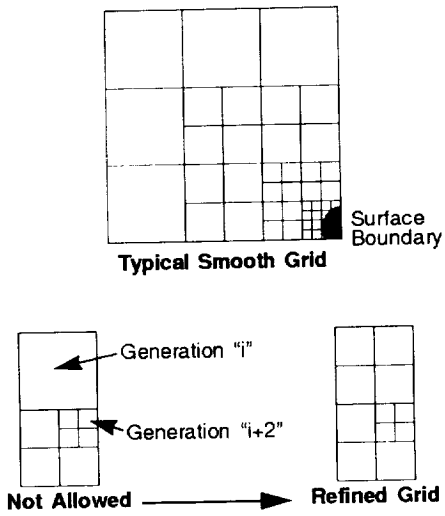



**Figure 1. SPLITFLOW cell development**

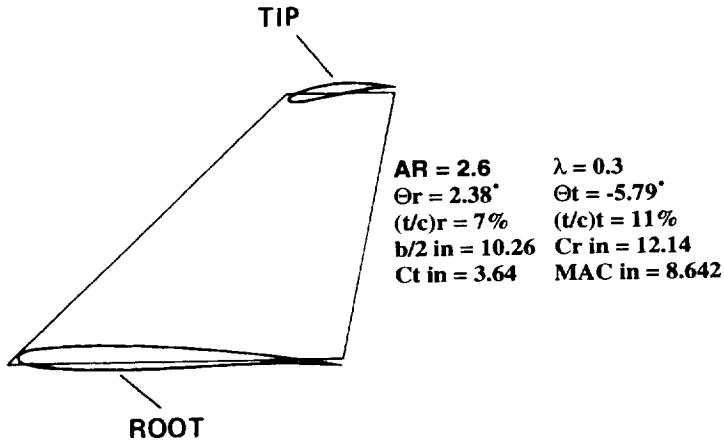



**Figure 2. Schematic representation of Wing C**

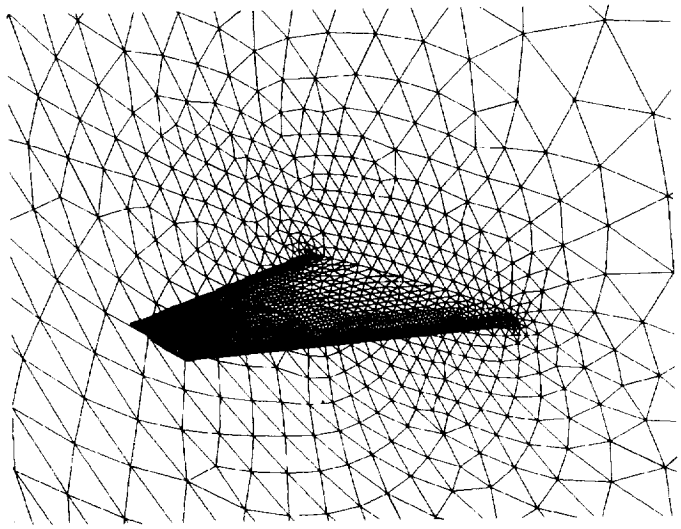


**Figure 3. Symmetry plane and wing surface triangularization from Vgrid**

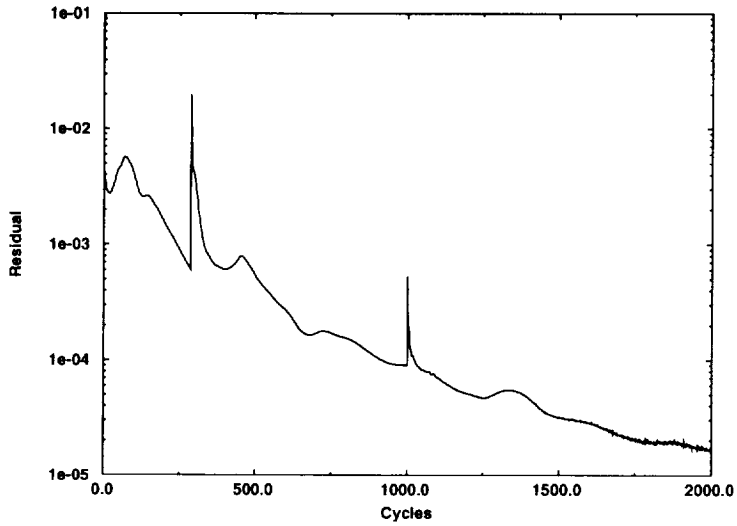



**Figure 4. Residual history from USM3D for Wing C test case**

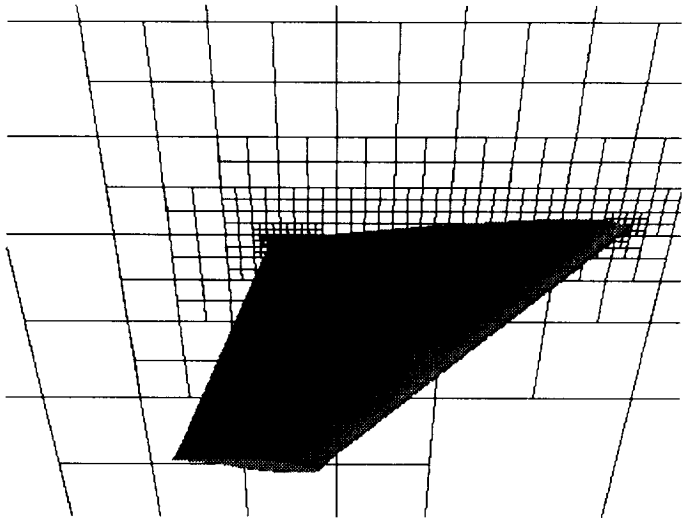


**Figure 5. Initial symmetry plane grid from SPLITFLOW**

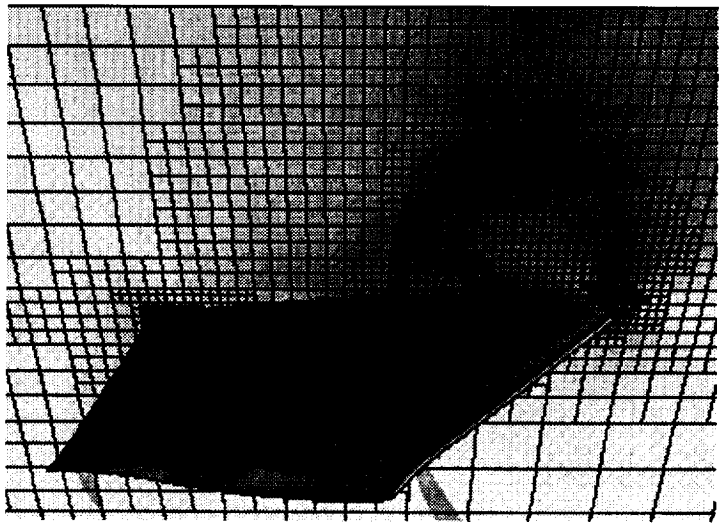
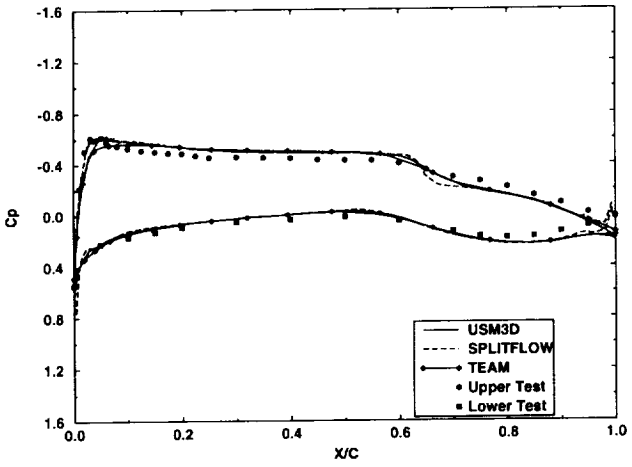
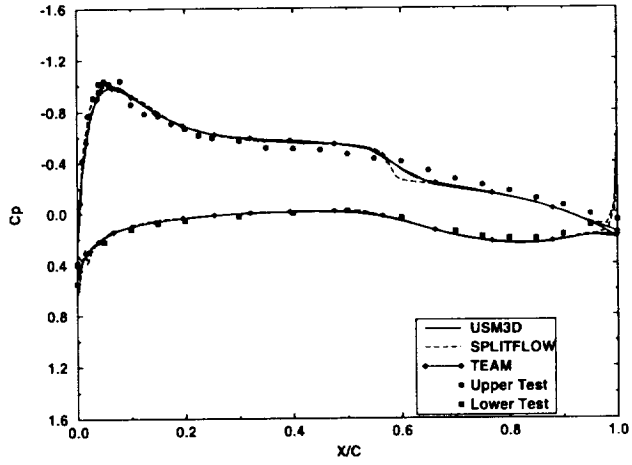


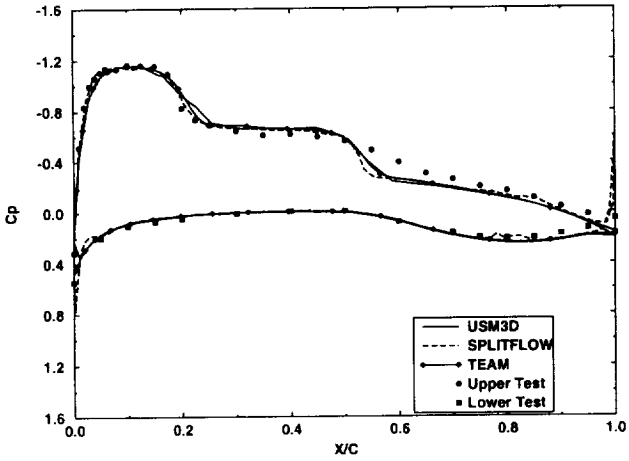
**Figure 6. Grid and solution from SPLITFLOW at the 70% span station**



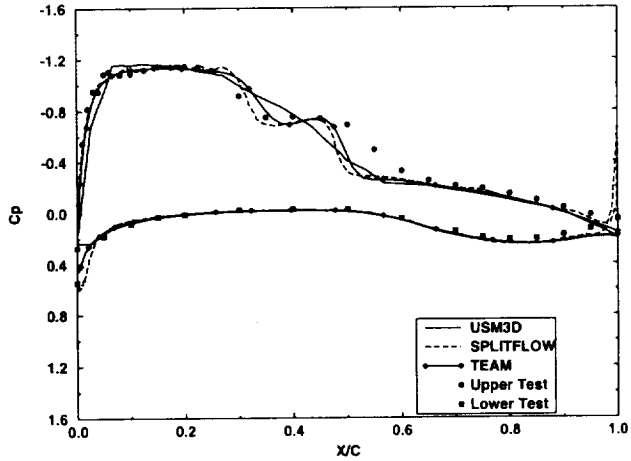

Eta = 0.1



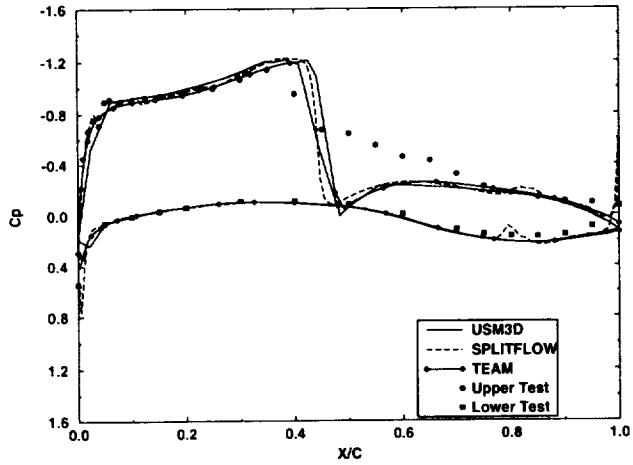

Eta = 0.3




Eta = 0.5




Eta = 0.7




Eta = 0.9

**Figure 7. Pressure distributions on Wing C at $M_\infty = 0.85$ and $\alpha = 5.0°$ using USM3D, SPLITFLOW, and TEAM compared to test data**

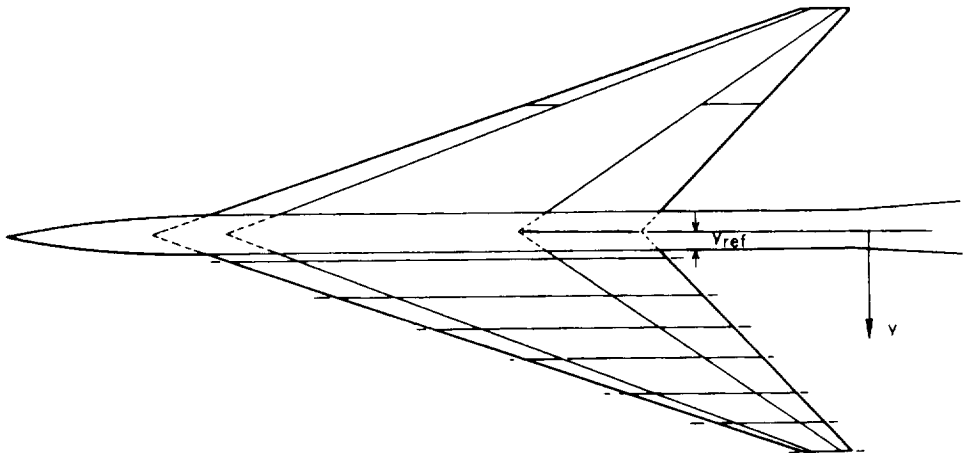



**Figure 8. Schematic representation of Arrow Wing Body configuration**

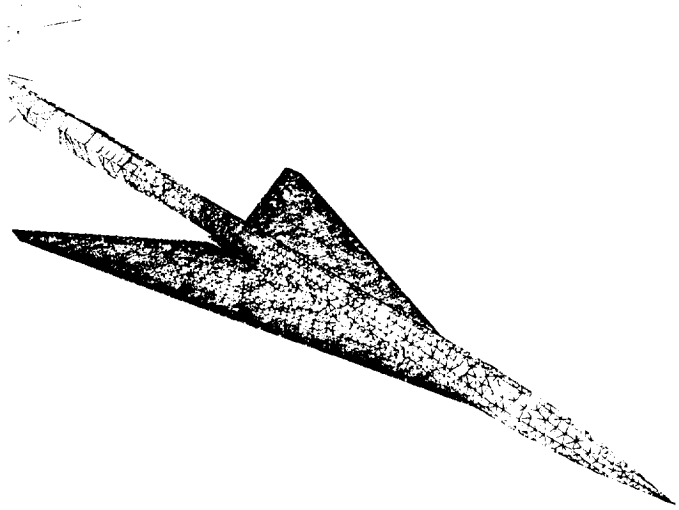


**Figure 9. Surface grid on Arrow Wing Body configuration from Vgrid**

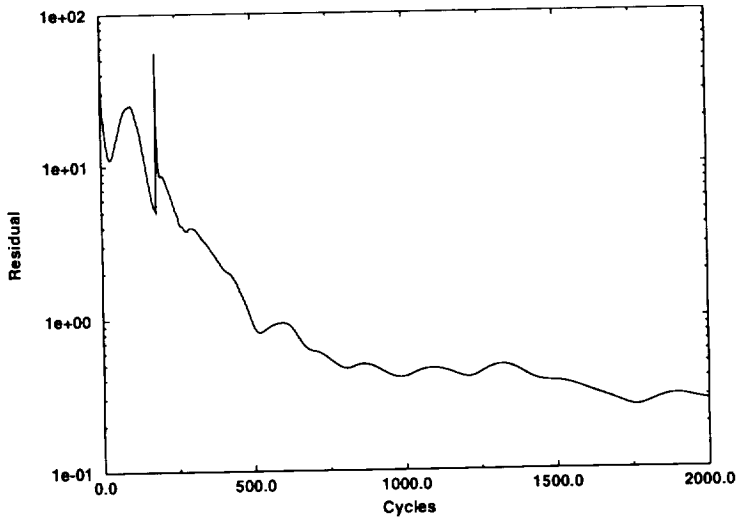



**Figure 10. Residual history from USM3D on Arrow Wing Body**

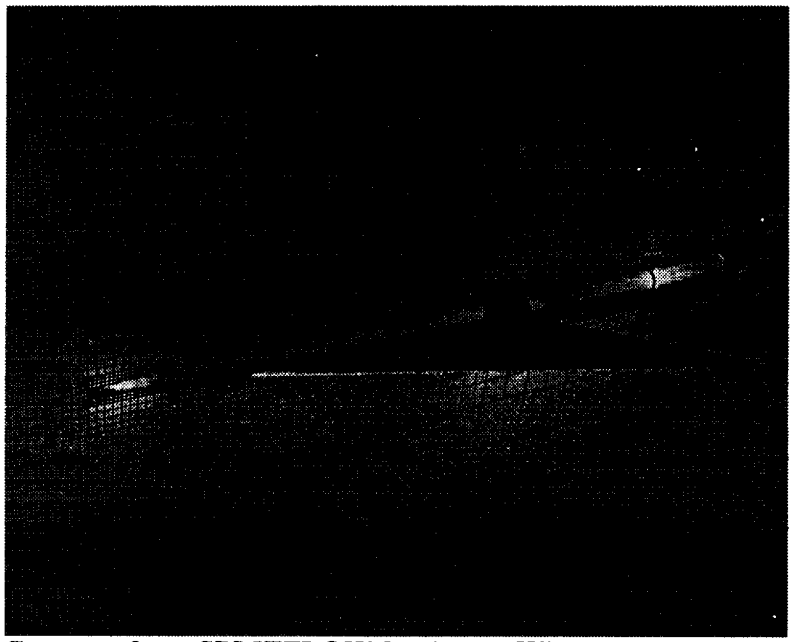


**Figure 11. Pressure Contours from SPLITFLOW for Arrow Wing Body case at $M_\infty = 0.85$ $\alpha = 4.0$ and $\delta_{TEF} = 8.3°$**

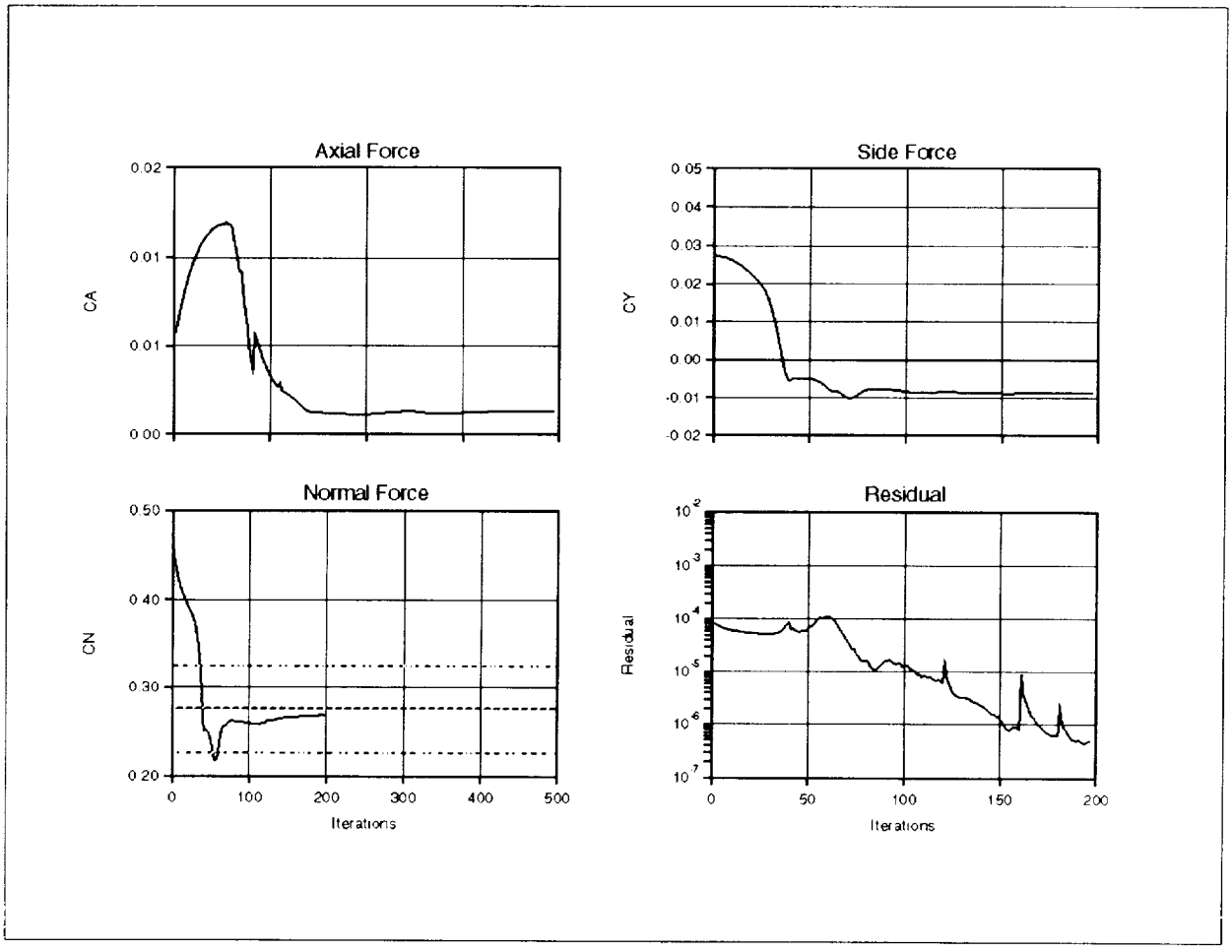
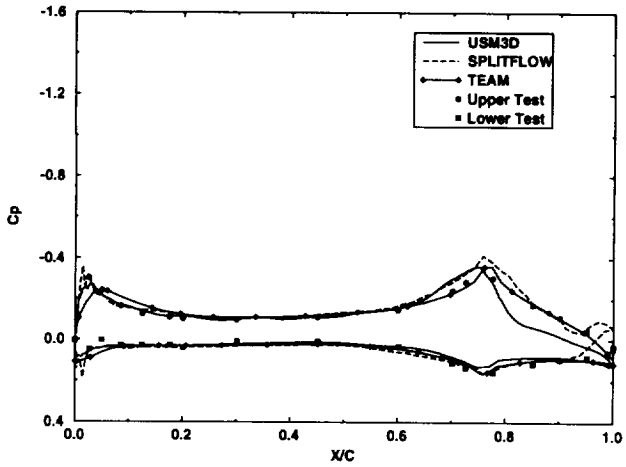
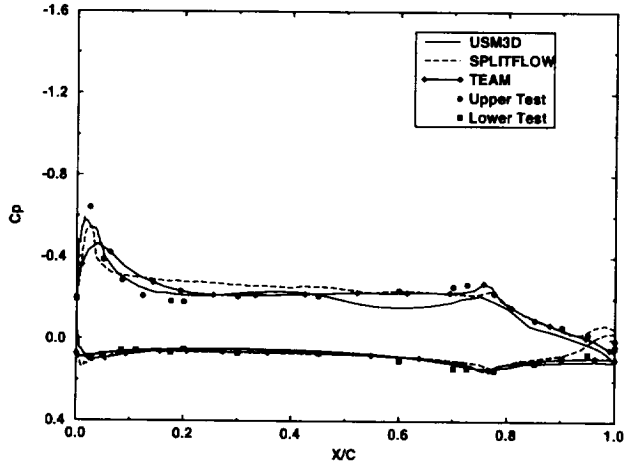


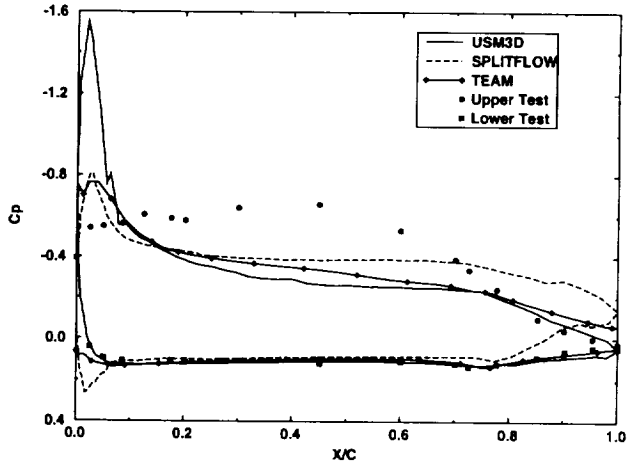

**Figure 12. Convergence data from SPLITFLOW for Arrow Wing Body case at $M_\infty = 0.85$ $\alpha = 4.0$ and $\delta_{TEF} = 8.3°$**




Eta = 0.2




Eta = 0.5




Eta = 0.8

**Figure 13. Pressure distributions on Arrow Wing Body at $M\infty = 0.85$ and $\alpha = 4.0°$ using USM3D, SPLITFLOW, and TEAM compared to test data**

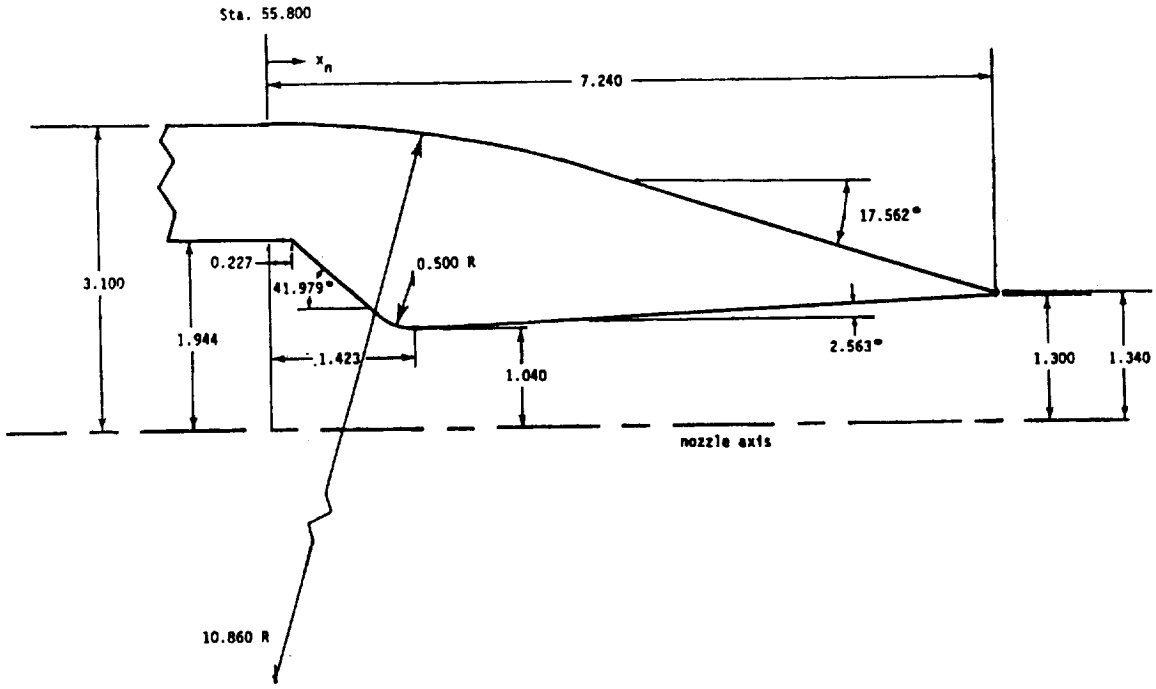



**Figure 14a. Top view schematic of NASA 2D-CD Nozzle**

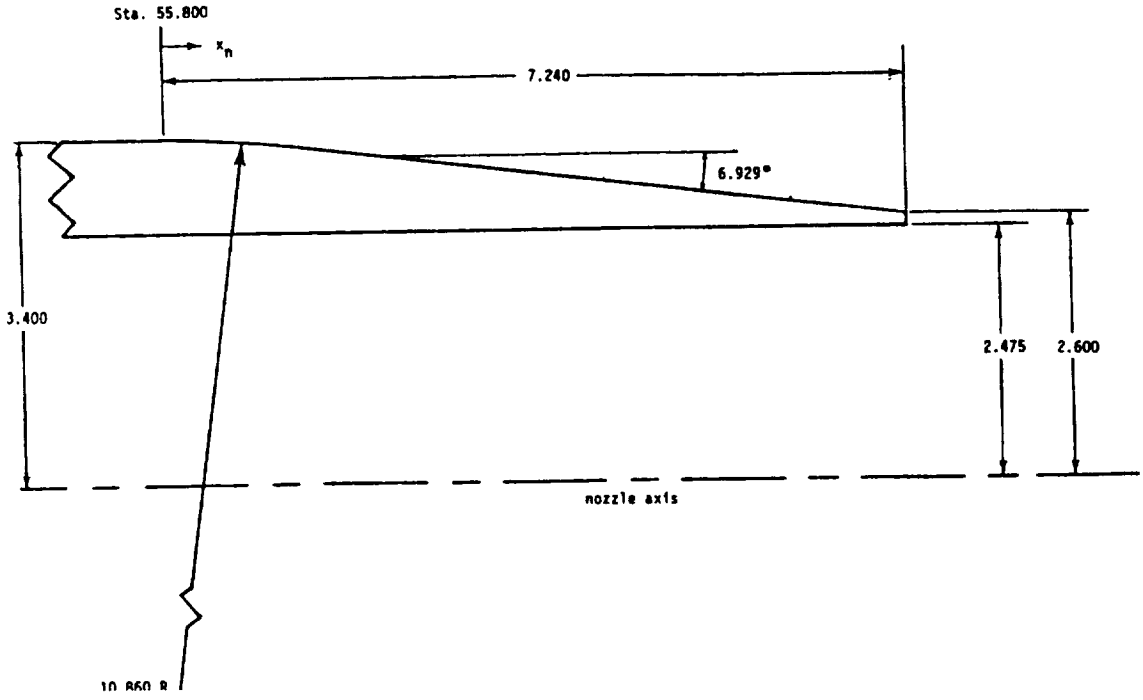



**Figure 14b. Side view schematic of NASA 2D-CD Nozzle**

N95-28748

# EVALUATION OF GRID GENERATION TECHNOLOGIES FROM AN APPLIED PERSPECTIVE

by
Gary S. Hufford, Vincent J. Harrand, Bhavin C. Patel and Curtis R. Mitchell
CFD Research Corporation
Huntsville, Alabama 35805

## ABSTRACT

An analysis of the grid generation process from the point of view of an applied CFD engineer is given. Issues addressed include geometric modeling, structured grid generation, unstructured grid generation, hybrid grid generation and use of virtual parts libraries in large parametric analysis projects. The analysis is geared towards comparing the effective turn around time for specific grid generation and CFD projects. The conclusion was made that a single grid generation methodology is not universally suited for all CFD applications due to both limitations in grid generation and flow solver technology. A new geometric modeling and grid generation tool, CFD-GEOM, is introduced to effectively integrate the geometric modeling process to the various grid generation methodologies including structured, unstructured, and hybrid procedures. The full integration of the geometric modeling and grid generation allows implementation of extremely efficient updating procedures, a necessary requirement for large parametric analysis projects. The concept of using virtual parts libraries in conjunction with hybrid grids for large parametric analysis projects is also introduced to improve the efficiency of the applied CFD engineer.

## 1. INTRODUCTION

Geometric modeling and Grid Generation are typically the most labor intensive tasks of a typical Computational Fluid Dynamics simulation. Factors which separate "good" tools from "bad" tools from the perspective of an applied CFD engineer include:

1. The ease/difficulty in obtaining a suitable surface definition for the particular grid generation scheme employed.

2. The ease/difficulty and the amount of time necessary to create an initial grid.

3. The ease/difficulty and the amount of time necessary to modify the initial grid both in terms of changing the number of grid points/distribution as well as the geometry, especially important for conducting parametric studies.

4. The amount of CPU required to run the Geometric Modeling/Grid Generation software as well as the amount of disk space required to store and re-create a model.

5. The applicability and efficiency of the CFD methodology for the particular grid created.

Each of these factors must be considered by the CFD engineer regardless of the type of grid generation/geometric modeling or CFD method employed.

In the commercial world the bottom line is the cost to complete a simulation. Cost manifests itself mainly in the form of man hours and computer equipment required. The overall time to complete a problem for a given methodology will always be related to the skill of the CFD engineer in using the available tools as well as the computer equipment available. Thus, evaluating and comparing methodologies can be quite subjective. Additionally "comparing methodologies" can often be confused with "comparing software tools." A poorly written software tool employing a given methodology tends not only to speak poorly of the software, but also of the methodology in general. This paper evaluates grid generation/geometric modeling technology in the areas of structured, unstructured and hybrid grids. The focus is on comparing the strengths and weaknesses of each methodology. Efficiency examples comparing methodologies are conducted using the current software tool, CFD-GEOM, developed at CFDRC which combines the geometric modeling and various grid generation methodologies in one integrated software package. The paper is divided into 8 additional sections: Geometric Modeling, Structured Grid Generation, Unstructured Grid Generation, Hybrid Grid Generation, Parts Libraries, Applied Comparison of Various Grid Generation Methodologies, CFD-GEOM: Current Status, and Conclusions.

## 2. GEOMETRIC MODELING

The definition of the surface for complicated geometric configurations is often one of the most laborious tasks in beginning the grid generation process. The surface definition is typically specified as a set of curves which interpolate to form a surface. For simple surfaces such as surfaces of revolution these interpolants are rather straightforward. However, for the generic three dimensional surface the problem is more difficult. For complicated surface definitions, most organizations use a CAD/CAM tool, which have the specific tools necessary for generating their geometries of interest.

The market is flooded with CAD tools used for design and manufacturing, however, the ability to transfer a CAD geometry to a surface geometry suitable for CFD grid generation is more limited. Two types of grid generation procedures are typical for CFD: structured and unstructured. The structured grid has been the historical mainstay of CFD while the unstructured grid is more common in structures applications. Specific strengths and weakness of each gridding procedure will be discussed later. However, the ability to transfer CAD geometries from any CAD/CAM system to a CFD grid generation system, either structured or unstructured, is crucial.

CAD tools can often be very specialized for specific applications and can be very large programs with hundreds of options. There is some attraction to incorporating a grid generation procedure into a CAD tool. However, in most CAD tools a significant amount of overhead makes this approach somewhat impractical due to memory requirements and CPU considerations. The closest tool to such a procedure is Icem-CFD[1], however, the incorporation into CAD is not direct. A transfer procedure from the CAD tool (DDN) to the grid generation tool (Mulcad or Hexa) is required. Other approaches use IGES[2] files as a starting point to create the grid. In this manner one can create

his geometry with any CAD tool and import the geometry into the grid generation system. As long as the grid generation tool can support CAD transfer without information loss this is a viable approach, although two tools must be learned by the CFD engineer. The point of view held at CFDRC is that the grid generation tool should support basic CAD-like geometry creation functions, while allowing users to import geometries from any CAD system adhering to industry wide CAD file protocols (such as IGES). In this manner a CFD engineer does not necessarily need to learn a complicated CAD package to construct and grid simple geometries, however, if necessary he may. Additionally the grid generation tool should be able to manipulate or clean a geometry created using a CAD package to make it more suitable for grid generation purposes.

To properly facilitate CAD data transfer without information loss the grid generation system must understand the language of CAD. In modern CAD systems this requires using NURBS (Non-Uniform Rational B-Splines)[3]. NURBS have become a CAD industry standard for geometric definitions in CAD/CAM and Computer Graphics. NURBS have replaced the methods of Bezier[4], Coons[5], and Gordon[6] surfaces. NURBS have gained widespread popularity in the CAD community mainly for their ability to represent a wide range of surface types exactly. For example NURBS can express Bezier, Coons, and Gordon surfaces exactly as well as conic sections, quadratic surfaces and surfaces of revolution.

The CAD industry has also become heavily involved in Solid Modeling and Parametric Design principles. Solid Modeling[7], a technique developed in the late 1970's and early 1980's, contains information about the closure and connectivity of volumes of solid shapes. It is becoming an increasingly important part of computer aided design of solid physical objects for design, analysis, manufacturing, simulation and other applications. The advantage of a solid modeling technique over surface modeling is it's ability to form closed and bounded objects more closely related to physically realizable shapes. Solid models can also distinguish between the outside and inside of a volume, thus allowing mass property analysis to determine volume, center of gravity etc.

Parametric design is a design system which allows the user to select a predefined set of geometric constraints which can be applied to the geometry being created. Such a system can determine the positions of geometric elements specified by the predefined combinations of these geometric constraints. This system may also have an equation solver which allows a set of engineering equations to be used to set the values of dimensions based on either engineering parameters or values of other dimensions. In general the parametric design system allows the user to change specific parameters or constraints and quickly, if not automatically, obtain a variation from the previous design. This system is in stark contrast to the traditional CAD system which requires manual specification of each geometric entity within a model with no linking between the entities.

From the applied grid generation perspective a user must be able to take his CAD model and generate a grid. This requires data exchange from the CAD to grid generation system. This is identical to the need to transfer models across different CAD packages. The CAD industry has developed several standards of data transfer to support data exchange across CAD systems. The Initial Graphics Exchange Specification (IGES)[2] was the first such standard. IGES can handle a wide range of geometric primitives including NURBS. Initial IGES releases do not support parametric

design or solid modeling information although these capabilities are planned in future releases. Other newer data exchange standards include PDES/STEP[8]. PDES (Product Data Exchange Specification) is an emerging standard for the exchange of product information among various manufacturing applications. The neutral exchange medium for PDES product models is the STEP (Standard for the Exchange of Product Model Data) physical file format.

The ability to begin with any geometric data (from any CAD source) and obtain a grid is a must for any grid generation system. When possible the parameters from the CAD package used in the design (ie. parametric design parameters, entity linking or solid modeling information) should be carried over from the CAD to grid generation system. The ability to perform this type of data transfer is only possible if industry wide standards are supported and adhered to by the CAD industry. As these standards develop they should be supported by the grid generation community. Currently NURBS are the highest level of surface grid generation supported by the standard CAD data transfer protocols and as such must be supported within the grid generation system.

## 3. STRUCTURED GRID GENERATION

The use of structured grids for CFD calculations is the historical mainstay of the CFD community. Viscous computations using structured grid methods are well established and the most advanced physical models are typically incorporated into structured codes. However, from the applied viewpoint, the grid structure often causes complications which are not physical in nature but are either geometric or related to the structure of the grid. This has led to unstructured grid CFD research where theoretically grid generation is easier. In practice the use of structured CFD grids, when used intelligently, can often be more efficient than using current unstructured techniques, especially when parametric analysis computations are the goal. This is possible since geometric information is inherently linked together in a structured grid through the grids structure. If small changes are made to the geometry (or grid distribution/density), the structured grid can be automatically and instantaneously updated if algebraic grid methods are used. In unstructured grids, these small changes require re-running of the unstructured grid generation procedure, a task which is trivial in terms of human labor but certainly not instantaneous. These types of issues will be the focus of the section entitled Applied Comparison of Various Grid Generation Methodologies later in this paper. In general a structured grid is defined as a grid which has a distinct i,j,k indexing. Four types of structured grid generation are typically used: *Algebraic, Elliptic, Hyperbolic and Advancing Layers/Advancing Normal.* Each of these techniques are most effective when object oriented multi-block grid generation is employed.

*Algebraic grid generation* is the most efficient means of grid generation in terms of computational speed and memory required. Algebraic grid methods rely on transfinite interpolation procedures to obtain a grid when the boundaries of the grid have been specified with a user-described distribution. The algebraic functions then interpolate to obtain a surface or volume grid. Many different interpolant procedures have been used to obtain a desired grid. These include the standard transfinite interpolant[9], Ericksson[10], Hermite orthogonal[11] among others with hybrid approaches blending the best features of each method. In all cases these methods are extremely fast (essentially instantaneous even on the smallest current workstations) and require very little memory. As such

very fine grids can be created very quickly. Typically these methods are used in a multi-block patched procedure. In this manner the CFD engineer de-composes the entire CFD domain of interest into patches (or blocks) and uses the algebraic methods for each block. He can then combine these blocks or keep them separate depending on the application. This method of grid generation requires more human effort than other methods to obtain an initial grid, however, the task can be made much easier with graphical CAD-like software tools. Once an initial grid is constructed, small modifications can be made and a new grid created instantaneously due to the geometric linking. The ability to instantaneously update a structured grid with minor modifications is dependent on the software design, however, it is possible and will be demonstrated in Section 7.

*Elliptic grid generation* was developed to "automatically" obtain a smooth internal grid distribution. Elliptic methods[12,13] rely on the solution of elliptic partial differential equations to obtain a smooth internal grid distribution. The user may specify controlling parameters which effect the grid clustering and grid orthogonality both internal and at the boundaries. Typically these methods work within a single patch or block. Thus, a user must still define the grid topology as with the algebraic methods, only now the solution of elliptic partial differential equations replaces the algebraic functions. The main advantage of this method over algebraic methods is it can provide for a smoother grid. However, in practice the user must play with the controlling parameters to obtain the grid he desires. Additionally elliptic techniques are much more computationally expensive and require much more memory usage than algebraic techniques. Thus, very fine grids can take a long period of time to create and require a relatively large amount of computer memory when compared to algebraic methods. In general the elliptic methods should only be used over an algebraic method when the algebraic functions are not providing the quality of smoothness desired or needed for the CFD solver. In practice this is a very small percentage of the time.

*Hyperbolic grid generation* is a grid generation technique based upon the solution of hyperbolic differential equations. Hyperbolic methods[14] were introduced to automatically generate a body-fitted orthogonal grid for external flow applications. The method uses an initial surface point distribution along with cell volume and orthogonality constraints to solve a set of hyperbolic differential equations to obtain the grid. In general the hyperbolic methods are one to two orders of magnitude savings in computer time over elliptic techniques[15] however, they are generally less robust than the elliptic techniques. They are also more restricted in the types of problems they can address since the outer boundary of the mesh cannot be specified. Currently the hyperbolic technique is generally used for Chimera applications[16] where the grid systems of several objects are allowed to overlap. For the hybrid grid generation techniques discussed below the hyperbolic system may be useful if a control is provided to prevent grid overlapping of several objects. This is analogous to setting a normal distance control. This requirement is the opposite of the Chimera philosophy to allow or enforce overset grids. In the hybrid system the gaps would then be filled with unstructured grid.

*Advancing Layer* or semi-structured grid generation[17,18] originated in the unstructured grid generation community. It is used to obtain viscous boundary layer grids near boundaries. Essentially the method "pops" a grid from a curve or surface in an automatic fashion based on the normal vectors emanating from the surface grid. The method does this in a layered fashion building

from the surface. If a structured quadrilateral surface grid is used the resulting grid is essentially a structured hexagonal grid near the surface, as such it is labelled in this paper as a structured grid method. The surface grid could also be an unstructured triangular grid resulting in a prismatic boundary layer grid that has a structure in the normal direction. Often times the near wall hexagonal grid is sub-divided to form tetrahedrons or the number of layers emanating from each normal vector is unequal resulting in an artificially generated unstructured grid. The true unstructured portion of the method results when this inner layer of semi-structured grid is merged with other unstructured grid methods in an "automatic" fashion. When quadrilateral surface grids are used this method is ideal for obtaining structured boundary layer grids very quickly. The debate about whether one should use prisms or hexahedrons in the boundary layer hinges on two questions. First, is it easier to obtain a controllable surface grid using a quadrilateral or triangular method, and second which grid type is more accurate and efficient in terms of the CFD solver. The use of NURBS for surface grid generation and its tensor product definition leaves the first question open while the second question is still being debated/reviewed in the flow solver community.

## 4. UNSTRUCTURED GRID GENERATION

A grid is said to be unstructured if the cells comprising the grid are not in any particular i,j,k order. Unstructured grids typically consist of triangles in 2-D and tetrahedrons in 3-D, although this does not have to be the case. Recently, unstructured discretization procedures have been extended to include prisms, hexahedra as well as combinations of these elements.

Unstructured grid generation has been dominated by three methods: Delaunay or Voronoi-based[19,20] techniques, modified quadtree/octree[21], and the Advancing Front[22] methods. Each of these approaches have been successfully applied for a variety of complex configurations. Delaunay methods require an initial distribution of nodes throughout the domain which are connected to form the unstructured grid. The points are connected to satisfy the Delaunay criteria to *promote* the generation of isotropic (or equi-angle) elements. In 2-D the elements are triangles while in 3-D the elements are tetrahedrons. By the very nature of the Delaunay criteria, the controlled creation of stretched cells in boundary layer regions is problematic. The Delaunay method also has the requirement that the points must be initially placed before they can be connected to form a grid. The modified quadtree/octree approach is another commonly used procedure which is based on subdivision algorithms to generate unstructured grids.

The Advancing Front Method is a method to both place points and create cells simultaneously. In the advancing front method the type of cells created can either by isotropic or stretched. The type of cells desired is specified on a background grid which encompasses the entire domain. Typically a set of Poisson equations is solved on the background grid[23] to control the cell size, cell stretching and cell stretching direction. In 2-D it has been shown that high fidelity stretched unstructured grids can be created with up to 20:1 base to height aspect ratios. In 3-D uni-directional stretching has been achieved.

The use of either Delaunay, octree or Advancing Front methods can obtain fairly smooth isotropic unstructured grids. Some general stretching can be achieved using Advancing Front

methodology, however, for extremely stretched cells necessary to resolve near wall effects and shear layers (1000:1 base to height ratio is typical) semi-structured[18] or layer[17] generation methods are used. The Advancing Layer/Advancing Normal method is typically exercised to refine regions of the domain which require highly stretched mesh features. The grid generation process then proceeds using either the advancing front or Delaunay method. This results in a completely unstructured grid with semi-structure mesh characteristics near boundaries. In general the unstructured grid generation methods are more computationally intensive than the structured grid generation techniques, especially as the cell numbers become large.

Theoretically the unstructured grid generation techniques require less manual labor than the structured techniques at a cost of CPU efficiency, i.e. algebraic vs. subdivision, subdomain removal, Voronoi-based. For parametric design purposes where many grids/geometries must be re-created over a period of time this trade-off may or may not be acceptable depending on the degree of the design change.

## 5. HYBRID GRID GENERATION

The term hybrid grid is defined in this paper as a grid which contains different grid types in different regions of a flow field domain. Generally the hybrid grid will contain a quadrilateral/hexahedral or prismatic grid in near wall boundary layer regions and a triangular/tetrahedral or adaptive cartesian grid[24] in inviscid regions. The appeal of hybrid grids is that more accurate and advanced structured flow solver methodologies can be used in viscous flow regions while the unstructured techniques can provide more flexibility in the gridding process. From the point of view of the applied CFD engineer, the hybrid grid gives much greater grid creation flexibility. When coupled to an object oriented grid generation software tool the CFD engineer can use different grid generation procedures in different flow regions utilizing the strengths of the different procedures and eliminating the weaknesses. The major drawback to using hybrid grids is that the CFD code must become more general, either handling structured and unstructured domains differently or handling multiple element grids in a purely unstructured fashion.

A significant advantage in using hybrid grids is that the grid can be separated into parts or objects. This is analogous to the multi-domain structured approach only now any grid type may be used in a particular domain. Since the flow field region is decomposed the individual domains can be modified as long as the interfaces remain unaffected. The use of hybrid grids is a relatively new development in the CFD world. It has enormous potential to speed up the turn around time for parametric analysis CFD computations when coupled to the concept of a parts library. The use of hybrid grids in conjunction with the parts library will be demonstrated in the next section.

## 6. PARTS LIBRARIES

An extremely important concept in the CAD design community is the use of Parts libraries. Typically, a CAD engineer will comprise individual parts of a design and piece them together to form the complete design. The concept of extending the parts library to include the grids is very attractive for many CFD analysis projects. This allows the CFD engineer to construct a grid around an object,

save the geometry and grid together in a library, and then piece the parts together in the final analysis project. This type of system is very suitable for hybrid grid generation. The near wall boundary layer grid can be stored with the geometry in a library, pieced together and then a general unstructured grid generation algorithm can be used to fill in the gaps. The Parts library concept is also extremely suitable in the Chimera method where the grids of each part are independent of one another, and can also be used in the patched structured grid method as well. The use of the parts library in the grid generation process is extremely suitable for parametric analysis.

The concepts of hybrid grid generation in conjunction with the use of Parts libraries for grid generation is very new. Therefore, an example of the power of these concepts will be demonstrated using a 3-element airfoil example. Figure 1 shows the base parts of the 3-element airfoil. Figure 2 shows two scaled versions of the slat and flap parts. Each of the individual parts is surrounded by a structured grid, Figure 3, and saved in a library. The outer boundaries for each of the slat and flap configurations, Figure 4, are identical so that they can be interchanged. Figure 5 shows the unstructured grid that fills in the gaps for the baseline configuration. An Advancing Front Technique with source controls[23] is used to obtain interface matching along the boundaries. Now if any of the other slat or flap configurations is desired they can be automatically interchanged without re-running the unstructured grid domain as shown in Figure 6. Additionally, for near wall O-grids, like that of the flap, if one wishes to refine the structured grid in the normal direction one can without any re-gridding of the unstructured part. In three dimensions where pure unstructured grid generators require a fair amount of CPU, this use of the hybrid grid generation concept along with the parts library concept saves a considerable amount of time for large parametric analysis projects. Additionally if the slat or flap angles of attack are changed or if the geometry is such that the outer boundaries cannot be efficiently made identical, only the re-solution of the unstructured domain is necessary. In most cases no new user input, only the geometrical change, is necessary when re-computing the unstructured domain. Figure 7 shows the grid resulting when the slat is pitched upward at 20 degrees.

## 7. APPLIED COMPARISON OF VARIOUS GRID GENERATION METHODOLOGIES

The grid generation methodology to use for a particular application will often be different depending on the goals of the project. Often times CFD engineers are limited by the type of solver they are using and by the types of grid generation tools available. The approach favored at CFDRC is to use Hybrid grid generation along with Parts Libraries to maximum the efficiency of the applied CFD engineer, especially for large parametric analysis projects. Each particular grid generation methodology has particular strengths and weaknesses. This section will look at the various strengths and weaknesses, with appropriate examples, of the following grid generation techniques: Algebraic structured multi-block, Elliptic, Unstructured Advancing Front, Advancing Layer/Advancing Front Unstructured, and Hybrid. When possible specific data (based on CFDRC experience) such as CPU time and human effort statistics will be given for each of the examples provided. As is always the case with human effort statistics they are very dependent on the user's experience level with a given methodology, and as such are somewhat subjective. The use of these statistics should not be taken as absolute but in relative comparison to one another.

Algebraic Structured Multi-Block

The strength of the algebraic multi-block methodology is its CPU speed and limited memory requirements. Very fine grids can be created very quickly. Additionally the ability to make small modifications and obtain instantaneous grid updates is possible (and demonstrated using CFD-GEOM) due to the linking between the geometry and the grid. The weaknesses of the algebraic multi-block methodology are the increased human effort to obtain an initial grid (ie. setting up the grid topology) and the possibility that the grid structure wastes cells. The example chosen for demonstration purposes is a grid of the B-1A Escape Capsule.

Figure 8 shows the multi-block structure of half of the B-1A capsule grid and the surface grid. There are 22 grid blocks to control the grid clustering which are combined to form a single composite block with blockages. The definition of the geometry was obtained from a Plot3D surface point set and broken up into the various topological domains in approximately 4 hours using CFD-GEOM. This includes creating additional geometry to form a topology and the topological refinement process to easily refine the grid at a later date. The grid consists of 90x75x41 grid lines comprising 263,440 cells. The final algebraic grid at the centerline is shown in Figure 9. Once the initial set-up is complete small modifications to the geometry or grid distribution are essentially instantaneous. For example using the CFD-GEOM software package the front spoiler can be rotated (Figure 10) to any angle and the corresponding geometry and topology which are effected is automatically updated in seconds (using an SGI Indy with R4600 processor). Although the initial time to set-up the topological structure is sometimes intimidating, an experienced user can use the advantages of the topology to easily and quickly modify the grid with instantaneous updating.

Elliptic

The strength of the elliptic method is the ability to obtain a smooth internal grid "automatically," reducing the need of the CFD engineer to pay quite so much attention to the grid topology. The weaknesses are that an initial grid is needed to begin the process (usually an algebraic grid), the CFD engineer must play with some clustering parameters to obtain a suitable grid, and the solution of the elliptic differential equations is CPU intensive and dependent on the grid size. An elliptic method has been applied to the algebraic grid of the B-1A Escape Capsule for comparative purposes.

Figure 11 shows the resulting elliptic grid obtained for the escape capsule. To converge the final elliptic result (using an in-house CFDRC elliptic grid solver using the methodology of Shieh[13]) takes approximately an hour on an IBM R6000 workstation. To obtain the proper clustering functions before obtaining the final result also takes some time. In general it takes 3 to 4 tries to find the proper clustering mechanism. In certain instances (for example at the capsule wing tip) the clustering functions and surface distributions need to be locally refined which takes some more time. For this particular case it took approximately 2 hours to obtain the proper clustering. Additional time is necessary to obtain the initial guess. In this case the final algebraic grid in Figure 9 was used, although if an elliptic method was desired from the start, the set-up time could probably have been reduced in half. The biggest disadvantage of this method is that to make a small geometry

change requires re-computation of the elliptic equations (another hour). If the grid is increased in size these computational requirements increase. If the size is increased by orders of magnitude computer memory requirements can also become an issue. Thus, this method becomes inefficient if a parametric analysis or grid refinement study is required.

Unstructured: Advancing Front

The strength of all unstructured techniques is that the applied CFD engineer does not have to set-up a grid topology, thus it is less user intensive. Once the geometry is defined the engineer specifies some controlling parameters and engages the unstructured grid method. The weaknesses of the unstructured method are the difficulties in controlling the resulting grid in both cell size and cell stretching primarily due to the difficulty in creating adequate background grids and developing controlling mechanisms for complex highly-curved geometries, and the high CPU and memory requirements for the algorithm. Progress has been made in 2-D cases for generic grid control in both cell size and stretching. Recent results show that stretching up to 20:1 aspect ratio[23] can be robustly achieved for arbitrarily complex geometries in the advancing front algorithm, however this is not nearly enough in boundary layers. In 3-D uni-directional stretching control has been achieved and generalized stretching for complex bodies is possible using the methods of Reference 23, however, sufficient stretching in boundary layers will be limited as in the 2-D case.

For comparative purposes an unstructured Euler type grid has been constructed for the B-1A escape capsule. The structured grids created for the capsule are somewhat stretched near the surface in the normal direction. For the unstructured grid this was not attempted. The resulting unstructured grid, shown in Figure 12, contains ~245k cells and took ~3.5 hours to complete on an SGI R4400 computer. Note that the grid is suitable only for Euler computations. With CFD-GEOM, the set-up time to define the capsule surfaces beginning with the Plot3D surface grid took approximately 5 minutes while the iterations to obtain the proper clustering took approximately 15 minutes. The strength of the method is the minimal set-up time, however, the CPU to obtain the grid is considerable. Additionally as the grid is refined the CPU requirement increases significantly and if the grid is refined by orders of magnitude the computer memory requirements become an issue.

Unstructured: Advancing Layer/Advancing Front

To combat the inability of the Delaunay or Advancing Front Methods to create a suitably stretched viscous grid, several researchers[17,18,25] have used an Advancing Layer or Advancing Normal approach to create semi-structured grids near boundaries and then merged this in an "automatic" way to an Advancing Front methodology when stretching requirements are reduced. The type of grid typically produced using this method is shown in Figure 13. Sometimes the quadrilateral near boundary elements are sub-divided to obtain a purely triangular grid.

The strength of this method, as in the pure Unstructured methodology, is the limited amount of user set-up. However, the computational requirements are still significant. In Advancing Front regions they are similar to the CPU requirements quoted above. In Advancing Layer regions the CPU requirement is less extensive, although the memory requirements are similar to the Advancing

Front requirements (much larger than for a structured grid) due to the indirect indexing of the created cells. The biggest weakness of such an approach is if small changes are made to the geometry or grid refinement is required the algorithm must be re-engaged, a significant computational expense if a parametric analysis is required.

Hybrid: Structured Algebraic/Unstructured Advancing Front

The use of purely Unstructured methodologies, such as the Advancing Front method, reduces greatly the amount of work a CFD engineer must do to obtain an initial grid. However, the speed and limited memory requirements with which algebraic (including Advancing Layer/Normal techniques) structured grids require and the speed with which the grids can be updated is extremely attractive for parametric analysis. Thus, the hybrid grid approach can take advantage of the best of the structured and unstructured methodologies. Additionally the hybrid grid method is ideally suited for the Parts Library concept described in Section 6. The near wall structured grids can be saved in a library with the geometry and pieced together. Then the automated unstructured methodology is used to fill in the gaps. A good example of this type of technique is given in Section 6, which demonstrates the advantages of the Hybrid grid methodology along with the Parts library concept.

## 8. CFD-GEOM: CURRENT STATUS

CFD-GEOM is an interactive geometric modeling and grid generation software package with fully integrated multi-block structured, unstructured and hybrid grid generation capabilities. CFD-GEOM is targeted towards the CFD end-user; the main goal is to enable a non-expert user (in geometric modeling and grid generation) to interactively create a moderately complex model relatively easy. CFD-GEOM uses the Motif graphical user interface environment in conjunction with the IRIS/GL and OPEN/GL graphics libraries to support compatibility across many different workstation platforms.

The **philosophy** of CFD-GEOM is that a single grid generation methodology is not suitable for all applications, as such different grid generation methodologies must be supported within the same software environment. In addition the geometric modeling capabilities are entirely based on the NURBS primitive allowing a CAD environment to develop geometry within the grid generation software. The focus of the overall CFD-GEOM software design is to **fully integrate** geometric modeling and various grid generation technologies into one data base. Thus, all geometric data, structured grid data, and unstructured grid data share a common data base and as such are completely linked together. The linking facilitates **automatic database updating** when any piece of the database is changed. These changes can include geometry changes or grid (density/distribution) changes. For example if a geometric entities shape is changed, immediately the effect of this change, and the number of other entities which are affected, is noticed throughout the rest of the database. Since the scope of the change in the database is known, only those parts of the database changed are updated. This facilitates the ability to update the minimum amount of information necessary. This communication throughout CFD-GEOM facilitates the instantaneous updating demonstrated in Figure 10. Additionally this software design allows for easy implementation of new technologies into the pre-designed data base.

The CFD-GEOM environment currently supports NURBS based geometric modeling capabilities, algebraic multi-block structured grid generation, multi-block unstructured advancing front grid generation, and hybrid multi-block structured/unstructured grid generation. Each of these functional operations is incorporated into a common database allowing changes of any kind to be felt immediately throughout the database. The functionality of each operation resembles separate software packages although they are fully integrated. For example, the geometric modeling capability of CFD-GEOM can be used as a simple CAD package irrespective of the grid generation capabilities and the grid generation capabilities, both structured and unstructured, can be used without the geometric modeling capability through IGES. Additionally a set of CFD-GEOM models can be stored into a virtual parts library and subsequently merged at will. This allows a user to create separate geometries and grids and mix and match them to form a new model. CFD-GEOM also filters all entity duplication between merged models, during IGES or Plot3D reading and at creation.

Specific geometric modeling capabilities within CFD-GEOM include: NURBS based toolkit to interactively define points, (poly) lines, arcs, free-from curves and surfaces; NURBS curve creation using arcs, splines, point-tangent, point revolution and point extrusion; NURBS surface creation using ruled, extruded and revolved surfaces; Interactive NURBS curve and surface modification by interactively modifying the control points and weights; Automatic intersection of curve and line sets including multiple intersections; Automatic intersection of surface sets up to all available surfaces simultaneously including multiple folded surfaces.

Specific topological and structured grid generation capabilities within CFD-GEOM include: Edge creation composed of single or multiple curves; Face creation from a set of 4 edge sets or directly on a NURBS surface; Block creation consisting of single or multiple faces sets; Block compositing to form a single composite block from a block set; Topology editing to re-orient a single block or to re-orient an entire connected block structure based on a single blocks orientation; Automatic grid updating when mesh density/distribution is changed; Edge Linking by point number, point distribution or both allowing the user to specify how changes propagate through the grid; Boundary condition identifiers to allow the flow solver to identify the location and type of boundary condition for a specified entity; Interactive definition/alteration of mesh point distribution; Multi-to-one connectivity for 2-D and 3-D blocks.

Specific unstructured and hybrid grid generation capabilities include: Automatic 2D/3D advancing front mesh generation; Source controls including point, line, curve, area and surface sources; automatic or manually controlled stretched cartesian background grids for grid control; General grid smoothing; Interfacing to structured mesh blocks for hybrid mesh generation.

Additionally certain operations can be completed on any entity including translation and rotation with automatic database updating, duplication and translation, and duplication and rotation. These functions in combination with the general curve and surface editor allow the user to make changes in the geometry (or add to the geometry) with immediate database updating, a capability specifically designed for large parametric analysis projects requiring a large number of small design changes.

Figure 14 shows a multiple, differing diameter pipe intersection created within the CFD-

GEOM environment. A key feature of CFD-GEOM is the ability to arbitrarily organize geometry and/or grid entities into parts. Each part can be arbitrarily manipulated and placed within any CFD-GEOM model. For example, the pipe section tear-off, shown in the lower left hand corner of Figure 14, can be rotated 180 degrees and glued back to the main model as shown. Other global part operations include translation, scaling, rotation and duplication. Additionally arbitrary shape modifications can be made by manipulating NURBS control points or ordered modifications such as face extrusion or rotation can be completed.

## 9. CONCLUSIONS

Based on comparisons between the various structured and unstructured grid generation methodologies it is apparent that one methodology is not universally suited for all problems. Couple this conclusion with the fact that a particular flow solver capability may only be available for a certain type of grid (ie. structured grid flow solvers are typically richer in physical models than unstructured flow solvers) and it is apparent that a single grid generation methodology is currently insufficient for CFD applications. Additionally, it is absolutely essential that the grid generation process incorporate advanced CAD geometric modeling capabilities to aid the CFD engineer in large parametric analysis projects. These realizations have led to the development of the CFD-GEOM geometric modeling and grid generation software system. The design of CFD-GEOM incorporates full integration between geometric modeling, structured grid generation, unstructured grid generation, and hybrid grid generation which allows for automatic and efficient database updating and the development of parts libraries. The software is specifically designed to aid the applied CFD engineer in large parametric CFD analysis projects.

## ACKNOWLEDGEMENTS

This work has been funded by the IR&D program of CFD Research Corporation. The authors would like to acknowledge Mr. Sami Habchi and Dr. Simon Ho for their help in generating the elliptic B-1A escape capsule grids, Drs. Ashok Singhal and Andrzej Przekwas for their support of this work, and Ms. J. Swann for preparation of the manuscript.

## 10. REFERENCES

1. Bertin, D., Lordon, J. and Marcaux, V., "A New Automatic Grid Generation Environment for CFD Applications," Aerospatiale, 1992.
2. Reed, K., Harrod, Jr., D. and Conroy, W., "The Initial Graphics Exchange Specification (IGES) Version 5.0," NISTIR 4412, NIST, Gaithersburg, MD, 1990.
3. Piegl, L and Tiller, W., "Curve and Surface Constructions Using Rational B-Splines," Comput. Aided Des., vol. 19, pp. 485-498, 1987.
4. Farin, G., Curves and Surfaces for Computer Aided Geometric Design, New York: Academic Press, 1988.
5. Coons, S., "Surfaces for Computer Aided Design," Technical Report MIT, 1964.
6. Gordon, W., "Free-Form Surface Interpolation through Curve Networks," Technical Report GMR-921, General Motors Research Laboratories, 1969.

7. Mantyla, M., An Introduction to Solid Modeling, Rockville, MD: Computer Science Press, 1988.
8. Wilson, P. and Kennicot, P., "ISO STEP Baseline Requirements Document (IPIM)," ISO TC184/SC4/WG1, Document N284, 1988.
9. Gordon, W.J., "Blending-Function Methods of Bi-Variate and Multi-Variate Interpolation and Approximation," SIAM Journal of Numerical Analysis, vol. 8, no. 1, pp. 158-177, March 1971.
10. Eriksson, L.E., "Generation of Boundary-Conforming Grids Around Wing-Body Configurations Using Transfinite Interpolation," AIAA Journal, vol. 20, no. 10, Oct. 1982.
11. Chawner, J.R. and Anderson, D.A., "Development of an Algebriac Grid Generation Method with Orthogonal and Clustering Control," Numerical Grid Generation in Computational Fluid Dynamics and Related Fields, A.S. Arcilla, J. Hauser, P.R. Eiseman, J.F. Thompson (editors), Eselvier Science Publishers B.V. (North-Holland), pp. 107-117, 1991.
12. Shieh, C.F., "Three-Dimensional Grid Generation Using Elliptic Equations with Direct Grid Distribution Control," AIAA-83-0448, AIAA 21st Aerospace Sciences Meeting, Reno, NV, Jan. 1983.
13. Thomas, P.D. and Middlecoff, "Direct Control of the Grid Point Distribution in Meshes Generated by Elliptic Equations," AIAA Journal, pp. 652-656, June 1980.
14. Steger, J.L. and Chaussee, D.S., "Generation of Body-Fitted Coordinates Using Hyperbolic Differential Equations," SIAM Journal Sci. Stat. Comput., vol. 1, no. 4, Dec. 1980.
15. Chan, W.M. and Steger, J.L., "A Generalized Scheme for Three-Dimensional Hyperbolic Grid Generation," AIAA-91-1588-CP, presented at 10th AIAA CFD Conference, June 1991.
16. Steger, D.L., Dougherty, F.C. and Benek, J.A., "A Chimera Grid Scheme," in Advances in Grid Generation, K.N. Ghia and U. Ghia, eds. ASME FED-5, pp. 59-69, 1983.
17. Pirzadeh, S., "Unstructured Viscous Grid Generation by Advancing Layers Method," AIAA-93-3453, AIAA 11th Applied Aerodynamics Conference, Aug. 1993.
18. Lohner, R., "Matching Semi-Structured and Unstructured Grids for Navier-Stokes Calculations," AIAA-93-3348-CP, 1993.
19. Meshkat, S., Ruppert, J. and Hungwen, L., "Three-Dimensional Grid Generation Based on Delaunay Tetrahedrization," Numerical Grid Generation in Computational FluidDynamics and Related Fields, A.S. Arcilla, J. Hauser, P.R. Eiseman, J.F. Thompson (editors), Eselvier Science Publishers B.V. (North-Holland), pp. 841-853, 1991.
20. Watson, D.F., "Computing the N-Dimensional Delaunay Tessellation with Application to Voronoi Polytopes," The Computer Journal, vol. 24, no. 2, pp. 167-172, 1981.
21. Yerry, M.A. and Shepard, M.S., "Automated Three-Dimensional Mesh Generation by the Modified Octree Technique," Int. J. Num. Method Eng., vol. 20, pp. 1965-1990, 1984.
22. Lohner, R. and Parikh, P., "Three-Dimensional Grid Generation by the Advancing Front Method," Int. J. Num. Meth. Fluids, vol. 8, pp. 1135-1149, 1988.
23. Hufford, G.S. and Mitchell, C.R., "The Generation of Hybrid and Unstructured Grids Using Curve and Area Sources," AIAA-95-015, Jan. 1995.
24. Coirier, W.J. and Powell, K.G., "A Cartesian Cell-Based Approach for Adaptively Refined Solutions of Euler and Navier-Stokes Equations," AIAA-95-0566, Jan. 1995.
25. Marcum, D., "Generation of Unstructured Grids for Viscous Flow Applications," AIAA-95-0212, Jan. 1995.

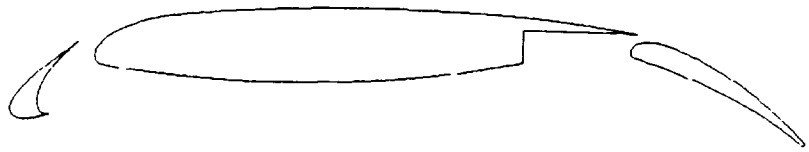


Figure 1. The base configuration of the 3-element airfoil case

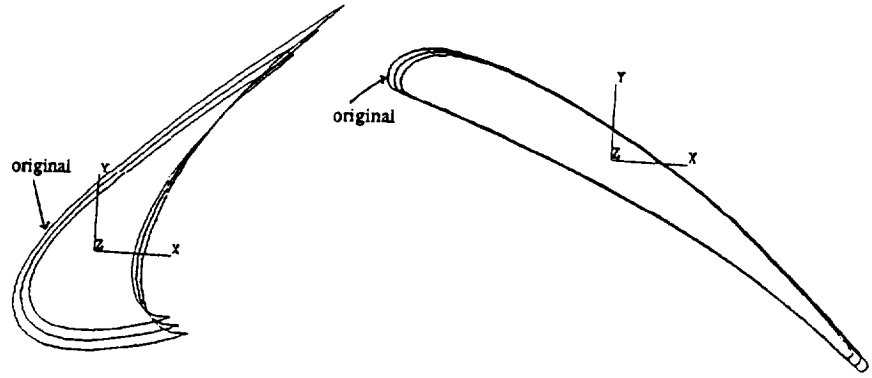



Figure 2. The scaled versions of the slat and flap parts compared to the original

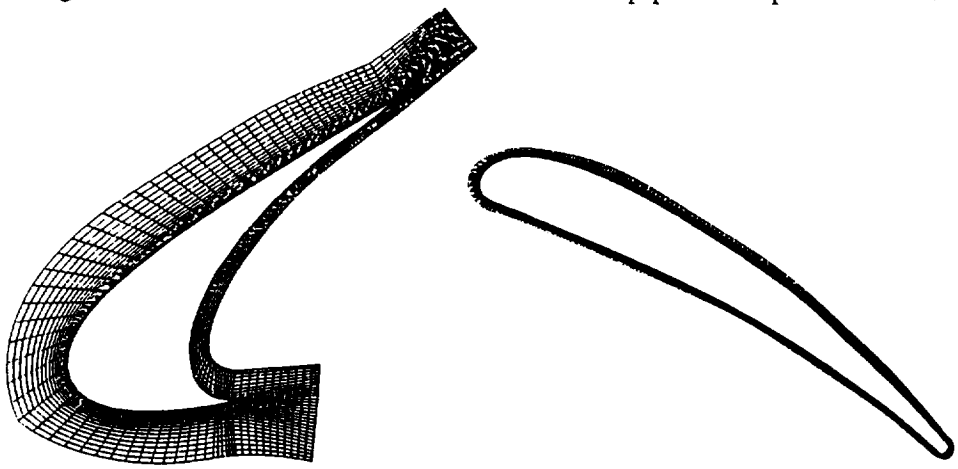


Figure 3. Structured grid surrounding the slat and flap parts. Both geometry and grid are saved in a library.

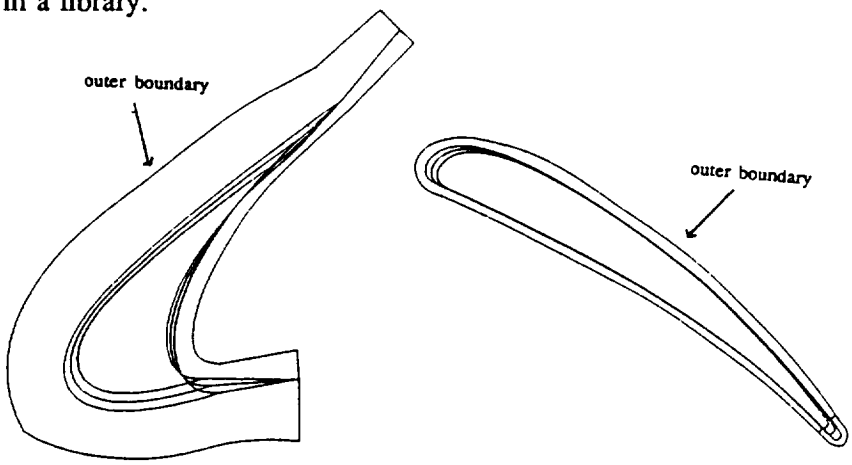



Figure 4. The common outer boundary for each series of slat/flap parts

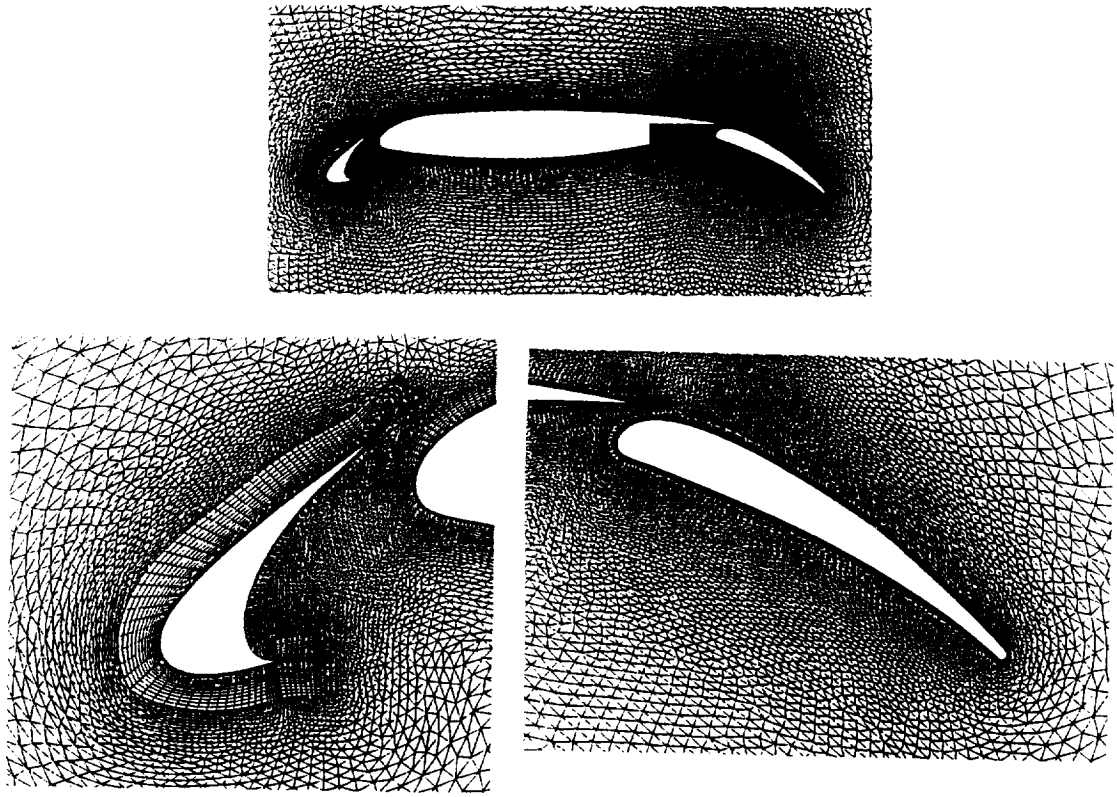


Figure 5. The unstructured grid that fills in the gaps between structured grid parts

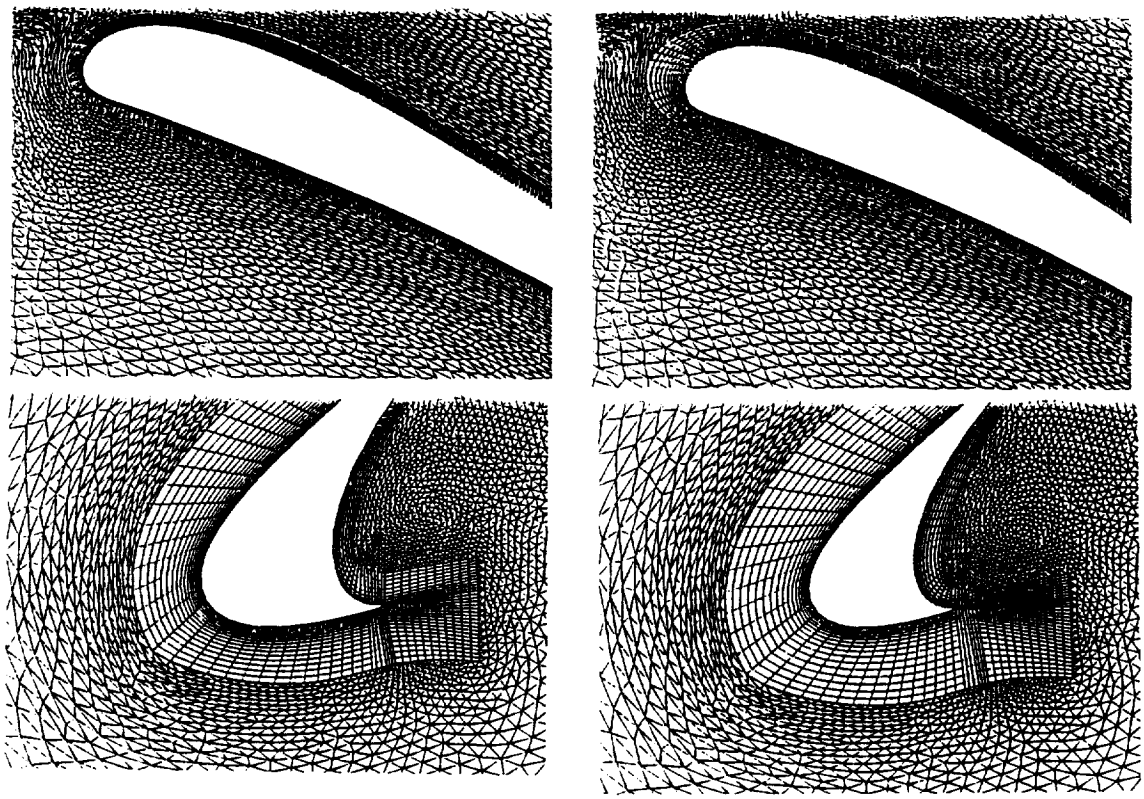


Figure 6. Interchanging of structured flap and slat grids without re-computation of unstructured grid

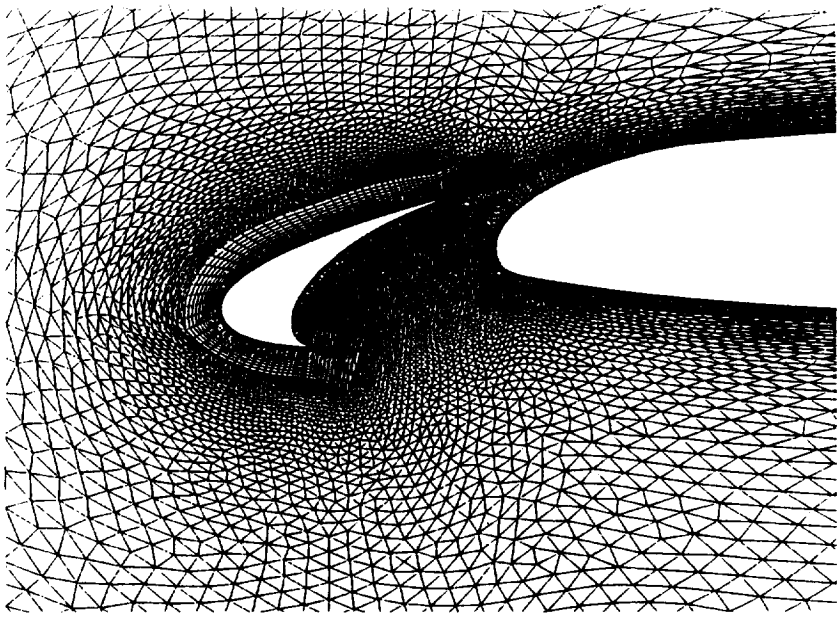


Figure 7. Example of slat rotation and unstructured grid re-computations with no additional user input

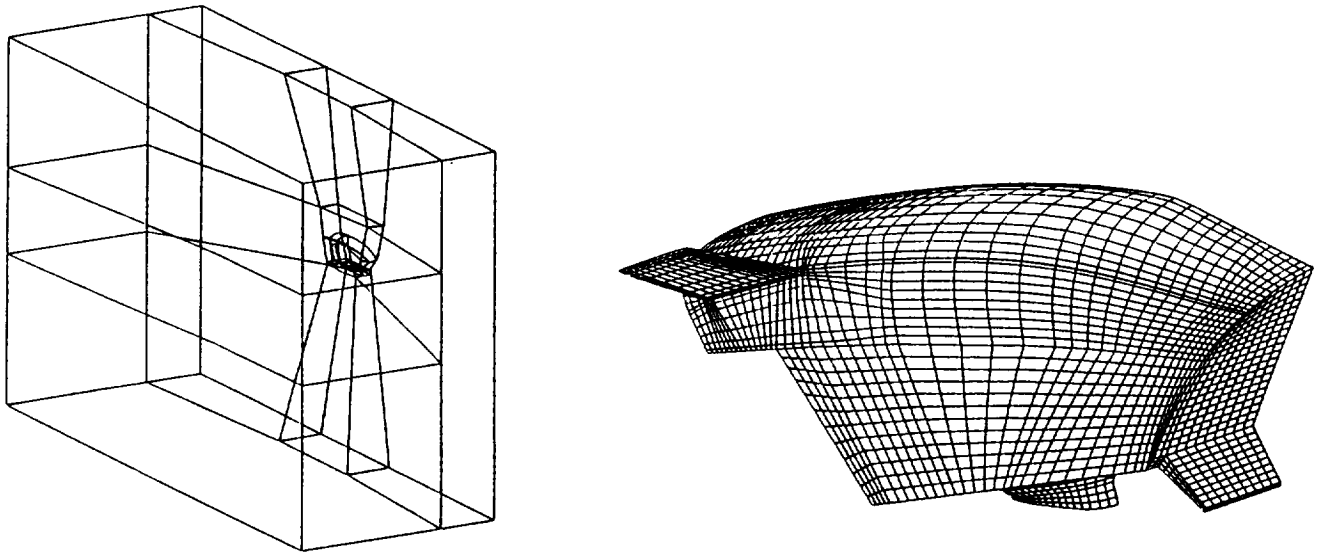


Figure 8. Multi-block topology of half of the B-1A escape capsule and the surface grid

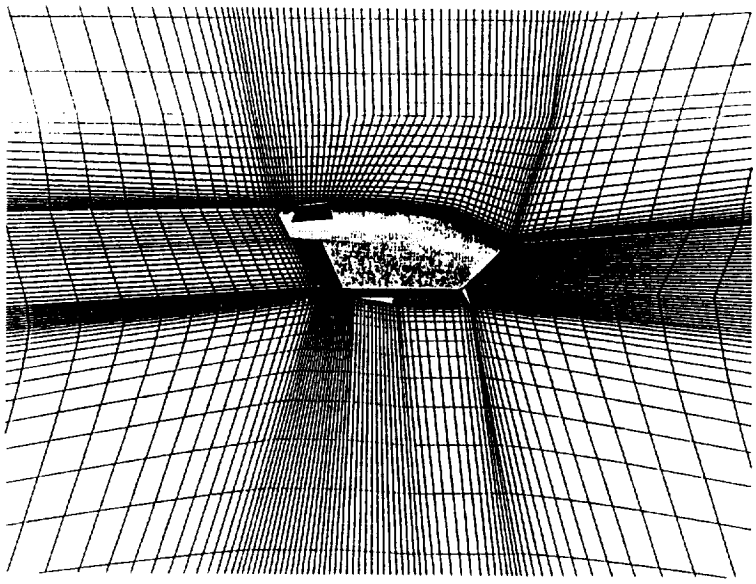


Figure 9. Algebraic grid of the B-1A capsule centerline

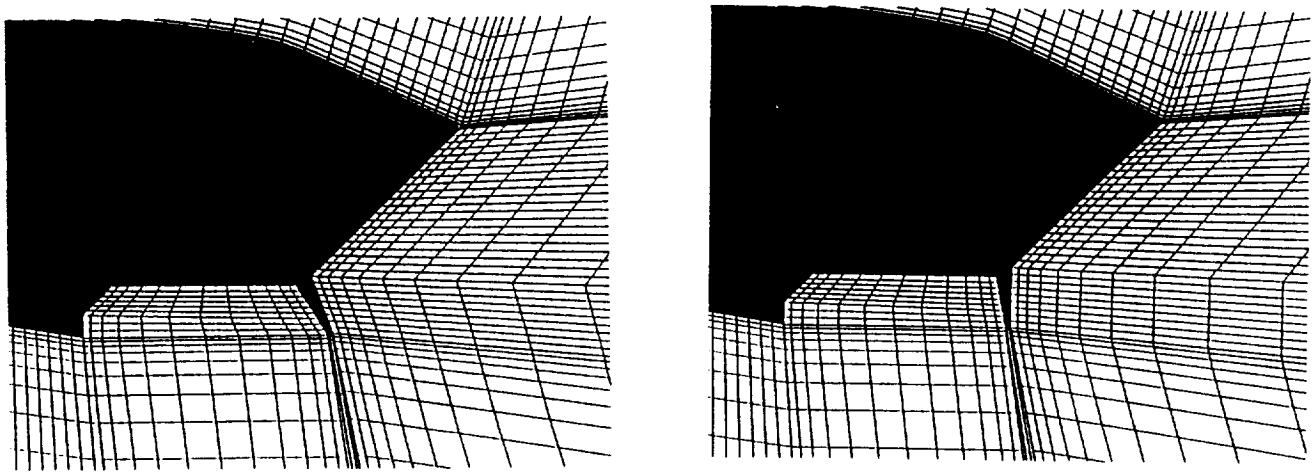


Figure 10. An example of a geometry modification (spoiler rotation) and immediate grid update using CFD-GEOM's structured gridding capabilities

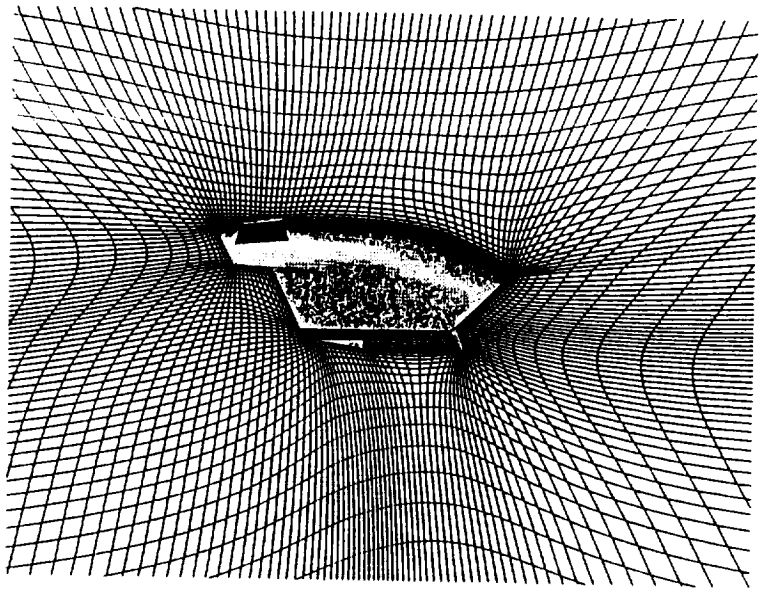


Figure 11. Elliptic grid at the B-1A capsule centerline

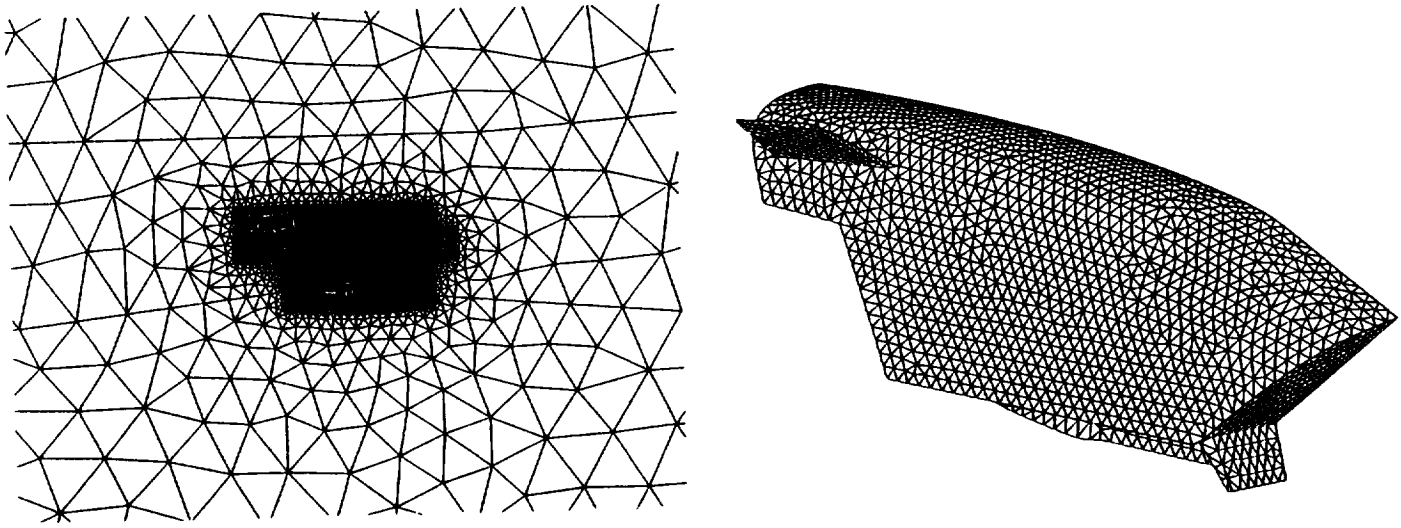


Figure 12. Unstructured grid of the B-1A escape capsule

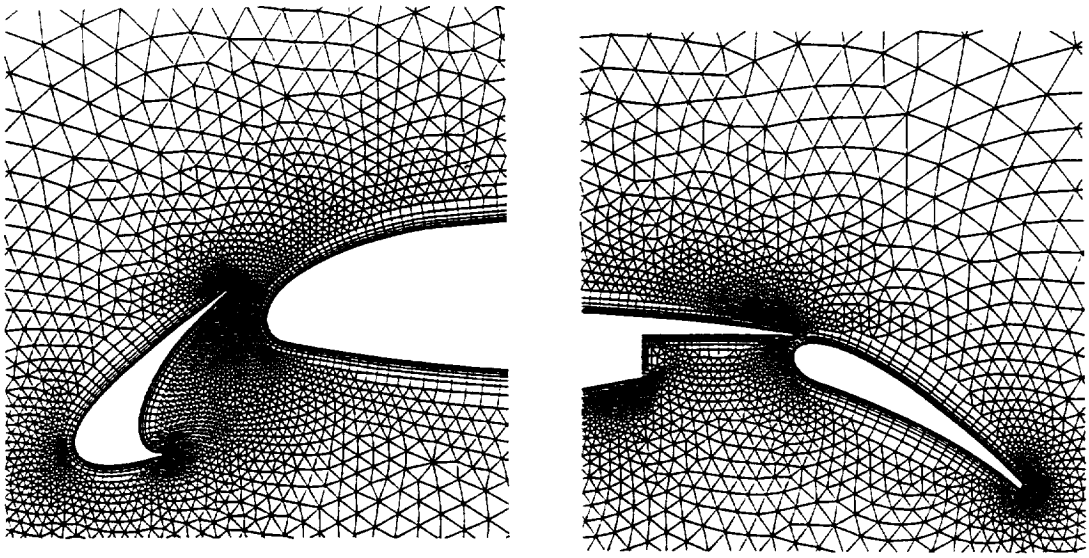


Figure 13. Type of grid produced from an advancing layer/normal type grid generator[25]

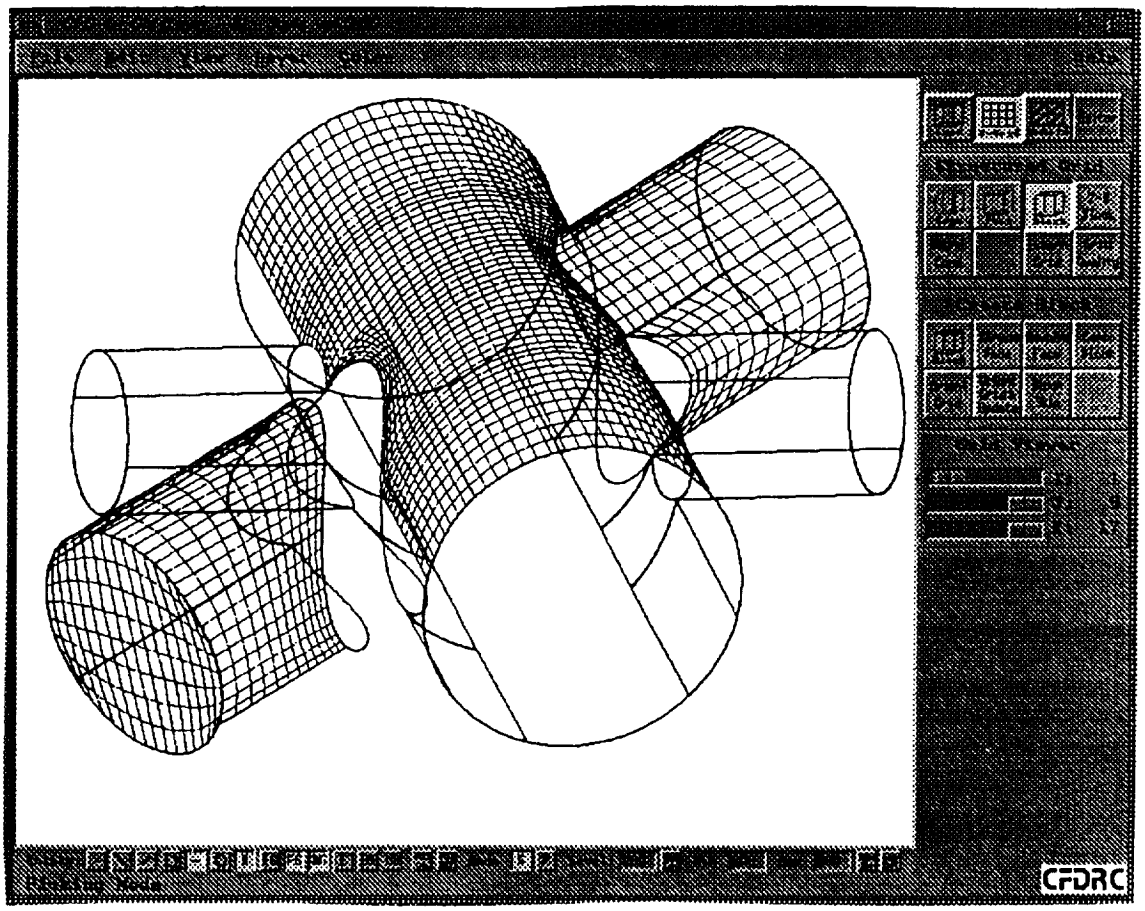



Figure 14. CFD-GEOM interface with a sample geometry and grid

# SOFTWARE SYSTEMS (1)

N95-28749

# THE NATIONAL GRID PROJECT: A SYSTEM OVERVIEW

Adam Gaither, Kelly Gaither, Brian Jean, Michael Remotigue, John Whitmire,
Bharat Soni, Joe Thompson
NSF Engineering Research Center for CFS
Mississippi State University, MS

John Dannenhoffer
United Technologies Research Center
East Hartford, CT

Nigel Weatherill
University of Wales
Swansea, UK

## INTRODUCTION

The National Grid Project (NGP) is a comprehensive numerical grid generation software system that is being developed at the National Science Foundation (NSF) Engineering Research Center (ERC) for Computational Field Simulation (CFS) at Mississippi State University (MSU). NGP is supported by a coalition of US industries and federal laboratories. The objective of the NGP is to significantly decrease the amount of time it takes to generate a numerical grid for complex geometries and to increase the quality of these grids to enable computational field simulations for applications in industry. [1]

A geometric configuration can be discretized into grids (or meshes) that have two fundamental forms: structured and unstructured. Structured grids are formed by intersecting curvilinear coordinate lines and are composed of quadrilateral (2D) and hexahedral (3D) logically rectangular cells. The connectivity of a structured grid provides for trivial identification of neighboring points by incrementing coordinate indices. Unstructured grids are composed of cells of any shape (commonly triangles, quadrilaterals, tetrahedra and hexahedra), but do not have trivial identification of neighbors by incrementing an index. For unstructured grids, a set of points and an associated connectivity table is generated to define unstructured cell shapes and neighboring points. Hybrid grids are a combination of structured grids and unstructured grids. Chimera (overset) grids are intersecting or overlapping structured grids.

The NGP system currently provides a user interface that integrates both 2D and 3D structured and unstructured grid generation, a solid modeling topology data mangement system, an internal Computer Aided Design (CAD) system based on Non-Uniform Rational B-Splines (NURBS), a journaling language, and a grid/solution visualization system.

## GRAPHICAL USER INTERFACE

Designing and implementing a consistent graphical user interface (GUI) is a key element in ensuring that the grid generation process runs smoothly and effectively. For this reason, a great deal of thought has been put into determining the requirements under which the GUI must perform, the design by which the requirements are maintained, and the implementation through which the technology may be transferred to the user.



PRECEDING PAGE BLANK NOT FILMED

## Requirements

Creating an environment for users to comfortably interact with technology is the key priority in creating a usable grid generation system. Ensuring that the environment is suitable follows from closely adhering to the following requirements:

- **Consistency** is perhaps the most critical requirement to maintain. This extends to the presentation of the technology as well as the look and feel of the GUI.
- **A logical organization of technology** has been maintained to present the information in a familiar manner to the user. This is accomplished by examining the function or process for which the GUI is designed, and organizing the process into tasks and subtasks.
- **Efficient use of screen real-estate** is an absolute necessity. Because current technology allows only a relatively small amount of two-dimensional screen space, careful consideration is given to the layout of the GUI.
- **Ease of use** is the facet of the GUI that ensures user acceptability or, more often than not, if the GUI is difficult or cumbersome to use, complete loss of information and inability to achieve user satisfaction.
- **Tailoring the system for user interaction** is a way to always keep in mind the technology that is being presented and the platform on which it is implemented.
- **Extendibility** of the system allows growth with both the technology and the intended user audience. It is a requirement from the initial design phase to allow for adaptability in every aspect of the system.

## Design

Having established the design requirements for the grid system, it is imperative to adhere to these requirements when completing the design phase. The design of the system can be thought of as consisting of six physical components and a seventh more abstract concept. The six physical components are the menu bar, the global actions, the global settings, the application palette, the message buffer, and the draw screen. A picture illustrating the layout of these physical components is shown in Figure 1.

The **menu bar** is designed to logically present the tasks and subtasks required to perform a high level process and can be found across the top of the GUI.

The **global actions** seen in the top right corner of the GUI are those top level functions that are required to complete the given task.

The **global settings** are designed to be information holders and are found in the space directly beneath the global actions. This global information consists of information that each task or subtask is going to require to complete the higher level process.

Expanded beneath the global settings is the **application palette**. Each task in the menu bar is considered an application, and when the user wishes to open an application, this is the space designated for its expansion. Expanding or opening a task allows it to become the active application. Because of the design of the system, only one application may be the active application at any one time. The application consists of actions and settings. The GUI is designed to be event driven, and therefore waits to be notified of an event that has occurred in the system. Each event in the system travels through one pipeline, and an event monitor polls the event at the end of this pipeline to determine which type it is. If the event is an application

event, the event monitor notifies the active application that an event has just occurred. The notification takes place through the use of a designated "handshake" routine. When the GUI is created, each application is forced to register a routine that will serve as its "handshake" routine. Once the GUI has given the event to the "handshake" routine, it no longer has control. It is then up to the application to decide how to proceed.

A **message buffer** is provided in the lower right corner of the GUI to provide feedback to the user. It is designated to be non-obtrusive, just giving the user a textual indication of the actions that are being performed and those that have been completed.

The **draw screen** makes up the remaining portion of the GUI. This is the space reserved for all graphical input and output. The draw screen is designed to be highly interactive allowing the user to interface with geometries and grids via the mouse and keyboard. All interaction between the user and the GUI must occur using either or both the global and applications actions and settings. The actions are designed to respond to a mouse press on the button, and to a keypress of the registered mnemonic for that action. The mnemonic is registered when the GUI is created and may not be changed. All invocations of action mnemonics must take place by typing the letter with the keyboard while the mouse is inside the draw screen. The size of the draw screen that is shown is what the user sees when the system is initially invoked. A full screen facility is designed to allow the user to get a larger view of the items that are being displayed. Interaction with the global and local actions is maintained while in full screen mode, but any user interaction with the global and local settings must be carried out when the screen is displayed in its regular size. This occurs because the global actions, global settings, application actions, application settings, and the message buffer are not seen when the draw screen is full size. The page up and page down keys on the keyboard are used to toggle between the regular screen size and full screen size.

As a more abstract concept, **user interaction** must also be addressed during the design phase. User interaction includes mouse interaction and graphical display. Careful consideration is made to allow for consistent behavior both in the mouse and in the interaction with the draw screen. The mouse is for the exclusive purpose of carrying out transformations and of acting as a pointer to a piece of information in the draw screen. Any textual interaction is completed through the combined use of the keyboard and settings. The graphical display is laid out in such a manner that user interaction is the primary focus. Several levels of display quality are provided to facilitate the need for user interaction and for the production of high quality pictures. These levels are maintained through the combined use of the resource file, and the view increase and decrease actions in the global actions space. The resource file is a user editable file that contains X Windows and Motif based descriptions of several components of the system. The user may either edit the file outside the system, or inside the system with the use of the resource editor. Several views may be set up and are each given a unique integer id. The views may consist of properties such as color, line thickness, display quality type, etc. These views may be displayed by typing the view number in the draw screen space and then pressing either the view increment or view decrement action. If there is no number in the numeric argument setting, the current view number is either incremented or decremented depending on the action that was invoked.

## Implementation

The NGP system employs the above design requirements, and is implemented in C using X Windows and Motif as the interface builder, and mixed-mode GL on the Silicon Graphics workstation as the graphics language. As OpenGL[2] becomes readily available, the system can be easily ported and then will become available for use on a variety of workstations.

## SOLID MODELING TOPOLOGY DATA STRUCTURE

The fundamental NGP data structure is called the Grid Topology Model (GTM)[3]. The GTM is based on a Boundary Representation (B-Rep) radial edge non-manifold solid modeling topology data structure.[4] B-Rep data structures are extensively used in current commercial CAD systems where geometric connectivities are an important aspect of geometric manipulations. The data structure provides explicit connectivity information between all geometric and grid entities, and abstracts the user from underlying geometric orientations. This abstraction provides a set of algorithms that simplify the grid generation process by allowing the user to ignore orientation when building multi-volume unstructured grids, multi-block structured grids, or when changing grid control parameters such as point distributions.[3]

The fundamental form of surface that is useful for grid generation is a non-manifold surface that can be accessed from N adjacent volumes and K adjacent surfaces. For structured grid generation, four-sided surfaces must be used. For geometry modeling and unstructured grid generation, multi-sided surfaces with holes (trimmed surfaces) can be used. The GTM provides the capability to read in trimmed surfaces and provide the same topological operations as four-sided surfaces. Figures 2 and 3 show the original geometry and the trimmed result.

The GTM is designed using a set of requirements formulated through interaction with the grid generation and CFS practitioners in the NGP consortium. The following is a list of the basic goals met by the GTM:

- **A foundation for general grid generation:** Includes 2D and 3D structured, unstructured, hybrid and chimera (overset).
- **No grid point duplication:** Grid points on shared geometries are not duplicated. Duplicating grid points causes wasted memory and holes or gaps between domains due to numerical inaccuracies.
- **Explicit adjacencies between volumes and surfaces:** Adjacencies between geometries and grids are explicitly defined. This enables multi-block point-point matching, point-point mismatching, full-face interfaces and partial-face interfaces. Access methods are available to provide adjacency, grid and orientation information to grid generators, elliptic smoothers, and CFS solvers.
- **Abstraction from geometric orientations:** The user does not have to keep track of block or surface orientations when building a grid.
- **Propagation of grid control information:** The user has the ability to propagate grid control information (number of points and point distributions) in a computational direction within a multi-block topology. The system also verifies that the propagated information is consistent throughout the grid.
- **Geometry verification:** The user is graphically prompted if the current geometry configuration is ready to be used for grid generation. Gaps, holes or overlaps where surfaces do not meet (within a user-specified tolerance) are indicated.
- **Semi-Automatic Boundary Detection (Blocking):** A set of algorithms detect bounded regions for both structured and unstructured grid generation.
- **Composite Edges and Faces:** Composite edges and faces allow for partial-face matching in multi-block structured grid generation.

A hierarchy of topology and geometry elements defines the classes of information contained in the GTM data structure. Each level of the hierarchy represents a level of abstraction between geometry and adjacency elements (Figure 4).[3]

All adjacent elements in the GTM are directly accessible via pointers (Figures 5 and 6). Searching is limited to traversing a doubly-linked circular list with length equal to the number of adjacent elements. Grid generation applications utilize the adjacency elements to store and access grid points on shared geometries. CAD utilities utilize the adjacency and geometry abstraction elements to determine degeneracies, orientations and adjacency. [3]

## GEOMETRY ENGINE

All geometry in the NGP system is represented by Non-Uniform Rational B-Splines (NURBS) [5-9]. Some of the reasons the NURBS representation is chosen for the geometrical database are as follows:

- **Unified Mathematical Model:** NURBS allow one mathematical form to represent both analytic and free form shapes. Thus the system can represent such shapes as conic sections, quadrics (cones, cylinders, spheres, etc.), and surfaces of revolution as well as free form sculpted surfaces with a single homogeneous database. The ability of NURBS to represent both analytic and free form surfaces with a single mathematical model is of particular importance because it simplifies coding of algorithms and reduces maintenance requirements on the geometry engine. A single data type can be used to represent all possible geometric entities within the system, and a single suite of evaluation and manipulation routines can be used to interact with them.
- **Generality:** NURBS offer the user flexibility to design a large number of shapes.
- **Efficiency:** NURBS evaluation algorithms are reasonably fast and numerically stable.
- **Geometric Interpretation:** NURBS can be interpreted geometrically. This allows designers with limited knowledge of the mathematics of curves and surfaces, but with a good knowledge of descriptive geometry, to create desired shapes with ease.
- **Local Control:** The property of local control allows relatively small areas of a curve or surface to be modified without affecting the entire spline.
- **Affine Invariance:** NURBS are invariant under affine, perspective, and parallel transformations (e.g. scaling, translation, rotation, shear, etc.).

### Geometry File I/O Capabilities

Exchange of geometry data with third party CAD systems is accomplished in NGP via IGES files which comply with IGES version 5.1 [10], but also includes the NASA IGES standard [11]. In addition to reading and writing IGES files in ASCII form, files compressed with the UNIX /bin/compress utility or the gzip utility can also be read. Other file formats which can be used for exchanging geometry information include Gridgen database files [12] and formatted, unformatted, and binary Plot3D [13] files. Discrete curve and surface data such as those in Plot3D files, Gridgen .dba files, and IGES entity 5001 are fitted with a linear or cubic NURBS spline [8].

### The Internal CAD System

Unlike other contemporary NURBS-based grid generation systems such as ICEM [14,15], NGP generates surface grids directly on the NURBS patches [16]. This feature allows surface grids to retain a high degree

of fidelity to the original geometry data; however, it also requires a high quality CAD model in order to produce good grids. The model should not contain any unwanted gaps, overlaps, or intersections between surfaces. The primary function of the internal CAD system is to allow the user to construct, modify and/or repair imported geometry in preparation for mesh generation. Models imported from traditional CAD systems generally require some degree of modification in order to make them suitable for use in the grid generator. The modifications may be as simple as deleting unwanted details or as complex as reconstructing large portions of the model. In either case, the CAD system provides the user with a rich set of tools to accomplish the task. In addition, many construction tools are provided which enable complex objects to be created from scratch.

The internal CAD system consists of nine individual applications grouped according to the functionality they contain. The buttons, toggles, and numeric fields within each application are arranged in a way which gives the user visual cues to functionality by grouping functions together with their input and/or option fields. The applications are as follows: **Points**, **Vectors**, **Curves 1**, **Curves 2**, **Surfaces**, **Blocks**, **Utilities**, **Edit**, and **GM Util**.

Point Operations—The point is used in construction of vectors, curves, and surfaces, to designate end points or control information. The point can be created in any application, but other useful operations are grouped in the **Points** application. These operations include averaging of points, total distance between points, angle between three points, and the closest point on a curve or surface.

Vector Operations—The vector is also used in construction of curves and surfaces and to designate slope, direction, or axis information. The vector can be created in any application by two points or a point and the components. The **Vectors** application allows the functions to calculate the normal and tangent vectors on a surface or the tangent vector for a curve. A vector direction can also be reversed.

Curve Operations—The CAD system provides both curve construction and modification/repair functions. The most used functions are located in the **Curves 1** application while lesser used creation functions are located in **Curves 2**. These curve functions consist of: 1) conic sections (circular arcs, circles, ellipses, parabolas, and hyperbolas). 2) spline curves (interpolation through points), 3) basic curves (lines, quadratic curves, and cubic curves), 4) averaging curves, 5) offset curves 6) splitting curves, 7) unioning curves, and 8) intersecting curves.

Surface Operations—It is generally necessary to modify an existing CAD model to correct for defects before it can be used for mesh generation. The functionality outlined below attempts to minimize the time and effort required to render a given set of surfaces usable for grid generation. Unlike the curve functions, all the surface construction functions are contained in a single **Surfaces** application. The following surface operations are available: 1) blending/ruled surface (two curves), 2) Transfinite Interpolation (TFI) surface (four curves), 3) degenerate TFI surface (three curves), 4) sweeping (one curve along another), 5) extruding (curve along a vector direction), 6) revolving a curve, 7) carpeting (projected TFI surface over a surface network)[17], 8) averaging, 9) extending or extrapolating, 10) intersecting surface with surface, 11) intersecting curve with surface, 12) splitting, 13) unioning, 14) smoothing (Smooths a surface according to curvature [5]), 15) reparameterizing, and 16) surface from cross sections.

Most of the above operations are exact in that the result is computed directly from the NURBS form of the given information. Exceptions to this are the carpet surface and surface from cross sections.

Utility Functions—The Utility functions are generic functions that can be performed on any NURBS curve or surface. These can be grouped into three categories: 1) transformations (translations, rotations,

scaling, and mirroring), 2) projection, and 3) extraction.

## AUTOMATIC BLOCKING

One of the most labor-intensive tasks associated with generating a block-structured grid is the creation of a framework of edges, faces, and blocks which together define what is known as the block structure. It is not uncommon for the generation of this block structure to consume well over half of the labor associated with any grid generation problem. Clearly, techniques for efficiently generating such block structures have the potential to increase the utility of block-structured grids into arenas which today are treated with overset or unstructured grids.

### Basic Approach

The basic idea behind the automatic generation of block structures is to shift the user paradigm from one which is *prescriptive* to one which is *descriptive*.

In a prescriptive process, the user has a set of low level tools which prescribe exactly *how* the problem is to be solved. Examples of these low level tools include drawing a line in space to serve as an edge of some block, generating an edge which is constrained to lie along a certain input surface, and connecting four edges into a face. While control at this level is certainly very powerful (and sometimes indispensable), it is clear to see that the labor associated with such an approach grows very rapidly, especially in configurations with multiple interacting components.

Alternatively, a descriptive process describes *what* the grid should look like. Example descriptions include the fact that the grid should wrap around the wing leading edge, that the inboard portion of the wing grid should lie on the fuselage, and that the store grid should lie within the wing grid. With this approach, the user could employ a divide-and-conquer approach, where the grid in the vicinity of each component is described separately, and the automatic blocker worries about interacting them to achieve an overall block-structured grid.

The key to being able to *describe* a block-structured grid is the development of a block-structuring *language* of some sort. In the current work, a graphical language composed of objects called cubes, wraps, attaches, holds, rulers, and spacers is used. (More details on the specifics of this language can be found in another paper in these proceedings [18].)

### Integration with NGP

As described above, the NGP system is composed of a large number of *applications* which execute independently, but which share the data which describes the configuration and grid. A key to integrating an automatic blocker into such a system is determining how and when the blocking procedure should be executed and on which data the blocker should operate.

An overall view of the grid generation process suffices to answer the first question:

- define the configuration, including any repair (to fill gaps, etc.) and augmentation (such as generating a far-field boundary) which is required to define a water-tight domain;
- develop a block structure, including specification of required number of grid points in each region, any user-specified spacings, etc.;

- generate the grid on the block structure, enforcing any grid controls specified above; and
- visualize and inspect the grid and either loop back to one the steps above to repair it or output it for use by a flow solver.

Clearly, an automated blocker should be executed after the configuration is defined but before the grid is generated.

The answer to the second question is not as straightforward. It is useful to think about two independent topologies within the grid generation process:

- the first is the topology of the configuration, which was essentially prescribed by the interrelationships of the surfaces and curves which define the configuration. The topological information contained here is generally of the form "the west edge of surface A is coincident with the north edge of surface B", or "curve C is formed by the intersections of surfaces D and E". This topological information is crucial for determining the structure and connectivity of the inputs.
- the second is the topology of the block structure, which prescribes the interrelationships of the edges, faces, and blocks which together define the block structure of the grid.

While there are certainly some correspondences between these two topologies, one in general does not want the configuration topology to constrain the block structure topology to any great degree. So in essence, the automatic blocker's job is start with an original (configuration) topology and to create a new (block structure) topology.

This is accomplished with the following procedure:

1. if the current topology is not a block-structure topology (that is, one generated in step 3 below by a previous execution of the automatic blocker),
   - then, save (in a file) a copy of the current (configuration) topology
   - otherwise, restore the (configuration) topology which was previously saved (in the file);
2. provide graphically-driven commands for the user to create the blocking objects (cubes, wraps, attaches, holds, rulers, and spacers) which describe the block structure;
3. create a new (block structure) topology within NGP which corresponds to the edges, faces, and blocks of the automatically-generated block structure; and
4. remove the original (configuration) topology from the system. Note that this step does not delete the curves and surfaces which define the configuration, but rather the topological information on which the block-structured grid generator will execute. (That is, we only want the block-structured grid generator to execute on the new (block structure) topology.)

Note that the first step, which involves the saving and restoring of the original (configuration) topology, is needed so that the automatic blocker always executes on a configuration topology.

One final note: it was stated above that control at the very detailed (low) level is sometimes indispensable for the generation of a suitable grid. This control can still be exercised within NGP by using the low-level

tools on the new (block structure) topology before the grid is actually generated; the only potential problem which one runs into when doing this is that these low level "tweaks" will be lost if the automatic blocker is re-executed (since is will again start with the "saved" configuration topology).

## STRUCTURED GRID GENERATION

The structured grid generation capabilities in NGP are based on the GTM data structure. This expedites the generation of both 2D and 3D systems. The use of the topological entities of the system (i.e. edges, faces, and blocks) to manage the grid information for each domain, allows the grid points to be stored once and be used on as many adjacent entities that share these common points. These shared points are determined by the topological creation of the faces and blocks in the domain.

### Topological Definition

All NURBS surfaces are automatically identified as topological faces. Other topological faces are created in a Right Handed System (RHS) from the wireframe description automatically, through the connectivity provided by the GTM data structure or the bounding curves can be manually selected. A recursive Depth First Search (DFS) is used to find all possible non-degenerate three and four curve cycles in a set of curves. These faces do not have any geometry associated with them other than which curves compose the boundary.

The topology-based blocking algorithm also uses the connectivity information provided by the GTM data structure to detect O-grid and H-grid blocks. A DFS algorithm is also used to find all possible three and four surface cycles that have valid end caps or the faces can be selected manually. Once a topological block has been detected, the surface orientations are flipped until a valid RHS is detected.

With the topological mapping of the edges, faces, and blocks, partial face matching is allowed by the construction of composite edges and faces. *Composite* edges are obtained by the concatenation of a series of edges into a single string of grid points, and *composite* faces are generated by the topological "linking" of the sub faces into a single consistent set of faces.

### Point Distributions

With the domain decomposition done in conjunction with the GTM, setting the number of points and setting spacing requirements is simplified by the propagation abilities implied by the data structure. The number of points applied to an edge of a face is automatically propagated to the opposing edge of the face. Through the connectivity the number of points are propagated to all other edges that lie in the same computational direction. Point distributions can be set on a single edge or propagated to a series of selected edges at the appropriate endpoints. The system also allows the user to specify interior distributions on an edge at selected points specified by the user, or by the detection of discontinuities. The topological mapping also allows spacings to be matched at vertices of connecting faces in a 2D system or for connecting blocks in a 3D system.

### Surface Grid Generation

Three types of surfaces are present in the grid generation system, NURBS surfaces, Four Curve Surfaces (FCS), and *Composite* Faces.

NURBS Surfaces—All meshes generated on NURBS surfaces are calculated in the parametric space of the underlying geometry to ensure the accuracy of the grid generation process. [16] Transfinite Interpolation (TFI) with arclength based interpolants is used to calculate the grid points in parametric space.

Four Curve Surface—TFI is used to generate the free form mesh for a FCS. This mesh can be projected onto a network of NURBS surfaces, thus creating a grid *carpet*.[17] Figure 7 depects a typical carpet grid spanning over surfaces patches of a shuttle canopy.

*Composite* Face—A *composite* face surface grid is created by the logical mapping of the grid points of the surface grids that compose the face into a single set of points.

Any modification of the edges automatically marks the adjacent faces as "dirty" and are regenerated. If any of the sub domains of the *composite* face are changed, the *composite* face surface grid is automatically updated to reflect the alteration.

## Volume Grid Generation

Volume grid generation is done on the blocks throughout the field using the 3D TFI routine. Each block is gridded individually, sharing common face, edge, and vertex points between blocks. Volume grids can also be generated through the use of Hypgen.

Hyperbolic Grid Generation—The hyperbolic grid generation algorithm is used to generate a 3D volume grid by marching away from an initial surface definition with a given initial step size and a stretching function in the normal direction. This is accomplished by solving the 3D hyperbolic grid generation system of equations (two orthogonality relations and one cell volume constraint).[19] Solving this system of equations allows for nearly orthogonal grid generation with excellent clustering control, and can generally be generated in orders of magnitude less computer time than elliptic methods. The method does not, however, allow outer boundary location to be precisely specified, and any discontinuities of the original surface tend to be propagated into the volume field.[20]

Hyperbolic grid generation is available in NGP through the use of the NASA Ames developed Hypgen hyperbolic grid generator. In cooperation with NASA Ames, Hypgen has been integrated into NGP as a subroutine to the overall structured grid generation. The system now allows for several hyperbolic grids to be generated at once, or several grids in series, if different control settings are required for each block. Once the hyperbolic grid has been generated, the information from the block is stored in the same data structure as the rest of the system, including the five faces, eight edges, and four vertices created by the hyperbolic grid.

## Elliptic Grid Smoothing

Elliptic smoothing is available in both the 2D and 3D systems in the form of surface and volume smoothers. FCSs are not smoothed as a single entity in the 3D system, but are smoothed in the volume smoothing. These faces are smoothed in two ways. First, if the face is shared by two blocks, a "sandwich" volume between the two blocks is created and that volume will be smoothed to keep the face updated with the interior. The second form of smoothing is performed if the FCS is a boundary face. The user sets the face to a sliding point condition, which will allow the points to float in accordance with the volume grid.

2D Elliptic Smoothing—To achieve slope continuity between regions, edges shared by two regions are smoothed by extracting the points for the edge and a strip of points from each adjacent face. Only the interior of the "sandwich" is smoothed, thereby adjusting the original edge to provide better continuity between faces. Vertices completely enclosed by regions are smoothed using a similar approach to receive continuity at the abutting of regions in the interior of the field grid.

Surface Elliptic Grid Smoothing—For 3D surface grids associated with NURBS surfaces, two smoothing

techniques are available. The first being an algebraic adaptive method to uniformly smooth the arclength distribution of the grid in the parametric space of the surface. This technique is usefully when a surface with an initially estranged parameterization is encountered (see Figure 8).

The second method of smoothing employs the elliptic solver on the parametric values of the NURBS Surface. Once the parametric grid has been adjusted by the elliptic solver, the parametric values are re-evaluated to give the new X,Y,Z points. Boundary conditions applicable for the NURBS surface smoothing are as follows: Fixed, Neumann Orthogonality, or Grape Orthogonality.

Volume Elliptic Grid Smoothing—To achieve slope continuity between blocks, faces shared by two blocks are smoothed by extracting the points for the face and a plane of points from each adjacent block. Only the interior of the "sandwich" is smoothed, thereby adjusting the original face to provide better continuity between faces. Edges and Vertices completely enclosed by blocks are smoothed using a similar approach to receive continuity at the abutting of blocks in the interior of the field grid.

## UNSTRUCTURED GRID GENERATION

Unstructured grid generation follows the same steps as the structured grid generation. First the boundary curves are specified, the surface grids are calculated, then the volume grids are calculated. The only difference between structured and unstructured is that the domain does not have to be decomposed. The user only has to determine a valid boundary of curves for 2D, or surfaces for 3D, prior to starting the grid generation process. In 2D, the user can manually pick boundaries and specify them, or use an automated feature that will detect the inner and outer boundaries of a configuration. In 3D, a set of fully connected surfaces is all that is needed.

Support is available for single and multiple domain 2D and 3D unstructured grid generation, parametric surface grid generation and multi-curve surface creation (where only curves are needed to build an unstructured 3D surface grid). Currently, both the Delaunay and Advancing Front methods are used for both surface and volume generation. These methods differ only in the point insertion criteria since both use the same boundary recovery technique as described in [21]. The points are automatically generated using the boundary point spacings to define the interior distribution and number of points. Boundary integrity is imposed by triangle and tetrahedra transformations. The implementation of the unstructured approaches are well documented in [21,22]. Figure 9 shows a Delaunay unstructured grid on region of the trimmed gasket. Figure 10 is an Advancing Front solution on the same region. Using these methods provide extremely efficient grid generation for very large problems. Most grids for super-computing scale solutions can be generated interactively on mid-range engineering workstations.

### 2D Grid Generation

2D unstructured grid generation entails that the user specify planar curves that form an outer boundary and any number of inner boundaries. Currently, these curves must be on the XY plane. Once the geometry is built, the user must build loops to identify the outer and inner boundaries (a single button push). Once the boundaries are identified, the user must indicate the number of points on each boundary, point distributions, point sources and/or curve sources in the field.

### Surface Grid Generation

To retain the accuracy of grid points on a surface, unstructured surface grids are calculated in the parametric space of the underlying geometry. [21] The generation of an unstructured grid on a surface is

defined by two independent variables U and V. This allows the grid to be computed in a square 2D space then mapped back into arbitrary 3D space by evaluating the UV values into XYZ values. This mapping is dependent on the shape of the surface and its underlying parameterization. For degenerate surfaces (a surface with one or more edges that collapses to a point) it is necessary to modify the shape of the parametric space.

Parametric Problems on Surface—Unfortunately, the parametric distribution on the underlying surface may not be optimal for grid generation. Currently NGP uses a mapping technique developed at McDonnell Douglas as part of their in-kind contribution to NGP. However, these mapping functions cannot solve all of the problems with parametric skewing in the grid. The ultimate answer (yet to be implemented) is to iterate between physical space and parametric space as it is done in the structured grid generation system.

## Sources

Point sources and curve sources control the size and density of grid points in a specified area or volume. Sources are available both on surfaces and in the volume. A source specified on a surface effects both the surface and the volume grid. The sources can be specified directly from a solver to enable grid adaptation through the paradigm. Point and curve sources are associated with vertex and edge topology elements respectively. A user-specified number of point sources can be placed on a curve (in parametric or physical space) in the same manner grid points are placed on an edge.

Source effects through boundaries—Currently, a source placed near a boundary has no effect on the boundary. Many problems are caused by this including rapid change in element size and extremely skewed elements. To offset some of the effects of the problem, boundary elements have priority over point source elements. This ensures that the boundary grid is consistent adjacent to geometry (for at least the first element width). However, this is only a temporary fix, the obvious answer to this problem is to let the effects of a source "pass through" the boundary. In 2D, the source would effect both boundary grids (edge grids) and any multi-domain regions on the other side of the boundary. In 3D, the source would effect edge grids, surface grids, and any other multi-domain region within the sphere of influence of the source.

## Multi-domain Unstructured Grids

In some problems it is necessary to define multiple regions, surfaces and/or curves within a field that must have grid points associated with it. Tools are available to the user that allow surfaces or curves in the interior of a grid domain to be marked as "transparent". This allows the grid generation algorithm to fix points on the original surface or curve, while building tetrahedra or triangles on both sides.

## Surface Defined by Arbitrary Curves in Space

Another feature of the 2D unstructured grid generation is the ability to build a 3D curved surface from a set of curves in space. This is extremely useful for capping open boundaries with an arbitrary surface definition where otherwise the original geometry would have to be cut to enable capping with four sided surfaces. Another useful feature for this type of surface creation is building planes of symmetry where a complicated geometry (such as an airplane) needs to only include half the geometry. Instead of making the outer boundary out of tens (or hundreds) of small surfaces, the user can simply define a few large surfaces for the outer boundary, and an arbitrary surface for the symmetry plane. The only restriction is that the multi-curve surface definition can only be created if the surface is singular (non-overlapping boundaries).

# VISUALIZATION

Over the past two decades, significant progress has been made in the levels of physical realism that can be incorporated in the numerical simulations of fluid flow. At the same time, slow but steady progress has been made in increasing the range of geometrical complexity which can be accomidated. This has been achieved through a combination of grid types, including block-structured, unstructured, and hybrid grids.

Currently the vast majority of labor hours in a typical CFD study is consumed in the grid generation phase. This is primarily because the closed-form solutions do not exist for flow fields around complex configurations and numerical techniques have to be used, thus requiring grid generation techniques to discretize the given domain. The process of generating a grid can be thought of as a set of iterations cycling through the following steps:

- **Specify:** The various "inputs" needed to generate a grid are prepared in this step. Examples of typically required inputs are grid topology, stretching factors, and numbers of points in various regions. By its very nature, this step is interactive.
- **Generate:** The actual grid points, their locations, and their interconnections are generated in this step. Because of its compute-intensive nature and its relatively long computation time, this step is usually performed in a batch mode.
- **Evaluate:** The user then needs to determine if the grid meets the requirements set by the expected flow physics and the flow solver which will be used. As with "specify", this step is interactive in nature.

Much effort has been expended in reducing the total time needed to generate grids. Some of the efforts has been centered on seamlessly integrating the above steps, resulting in the production of grid generation systems such as EAGLEView [24], GRIDGEN [12], GRAPEVINE [25], and ICEM [14,15].

Current grid evaluation techniques fall into two groups. The first involves the stand-alone computation of "grid-quality" measures, allowing the user to assess the local goodness of the grid. The second set of grid evaluation techniques is aimed at producing visual representations of the grid and possibly its quality measures. Unfortunately, to date insufficient effort has been focused on techniques for the rapid and effective *interactive* evaluation of grids in their various stages of generation. Such interactivity is essential for providing information about the grids in a way which can enable the user to correct or improve the grid with the available grid tools.

Hence, the goal of the present grid visualization system is two-fold: first, to provide a suite of techniques for effectively assessing the quality of block-structured, unstructured, and hybrid grids; and second, to ensure that the techniques can be used during the grid generation process as an aid in assessing the validity of the operations performed thus far.[26]

## Design Considerations

The design of the techniques presented here was driven by two considerations: what questions would the user like the system to answer, and which visualization tools convey the answers to those questions most succinctly.

The questions which the user is likely to ask include:

- Has the grid been generated in the correct region in space?
- Are there parts of the grid where the grid points are either unacceptably close to or far from each other?
- Are there local areas of the grid which have problems? If so, where are they and more specifically, what are the problems?
- What are the causes of the problem(s) and how can it (they) be fixed?
- What is the overall quality of the grid and is it valid to progress to the next phase (i.e., surface grid to volume grid to field solution)?

The selection of an appropriate set of visualization tools was guided by examining those used in other systems and identifying weaknesses. The major shortcomings identified included:

- **Information overload:** Many grid systems are not discriminating about what is displayed. Wire frame displays of grids, even those which employ advanced techniques such as depth-cueing and motion, are extremely hard to decipher.
- **Information deficit:** The current systems do not "point out" to the user the problems which exist in a grid. Many CFD projects have been sabotaged by bad grids which did not get discovered until after the flow solver engineer labored intensively over the problem.
- **Lack of context:** There are really three problems here. First, most current techniques do not give the user an effective frame of reference; once the user is alerted to the problem, it is hard to determine exactly where the problem is and how it interacts with the configuration. Second, they do not show the interrelationships between the various grid quality measures which may be locally important. And third, the techniques which are used separate the display from the discrete nature of the grid cell data. Iso-surface plots of grid quality only convey that there is a problem, but not why there is a problem.
- **Difficult to learn:** Because most systems are nonintuitive, they have an extremely long learning curve.

## Visualization Hierarchy

Analysis of the above questions shows them to be hierarchical in nature, thus leading naturally to a set of visualization tools which are also hierarchical. The following sections describe a specific hierarchy of grid visualization and analysis tools used to not only demonstrate the quality of a grid, but to allow the user to specifically isolate the area or areas of the grid that are causing problems. Making up the hierarchical tree of evaluation are qualitative tools which allow individuals to determine the basic overall quality of a grid by inspection, quantitative methods to flag areas of numerical interest, and grid browsing techniques to step into and through the grid and view the areas of interest. This visualization hierarchy is intended to act as a three-stage process for determining the quality of a grid, with each level portraying a more localized and detailed set of information.

Qualitative Evaluation—The qualitative evaluation stage is intended to assess the overall placement of grid points, both for block-structured and unstructured grids. Since at this initial stage in the analysis, only grid points (and not their connections) are important, a visualization tool called a *point cloud* was developed. This visual display is created by placing a small point on the screen at each grid node. An example of an unstructured *point cloud* is shown in Figure 11.

The simplicity of this display technique offers two distinct advantages. First, because of the absence of grid lines, the screen is relatively uncluttered, especially as compared with previous visualization techniques; this is very important for unstructured grids which heretofore have proven difficult to visualize. Second, the interactive speed of pans, zooms, and rotates is high enough so as to make "motion" of the display an effective means of extracting the three-dimensional nature of the grid.

Because this technique is fast, and because it has been included "on line" within the NGP system, any deficiencies which are discovered with this tool should be fixed before continuing on with the grid generation process.

Quantitative Evaluation—The next, more in depth, evaluation stage is quantitative evaluation. Here, the objective is to elicit from the system quantitative measures of grid quality. Of course, there is no single measure of quality but rather a set of mutually interacting measures such as skew and stretch. The relative importance of each of these measures can only be assessed in the context of the flow solver which is to be used.

The need to present complex, mutually-interacting data is not unique to grid visualization, or even to scientific visualization as a whole. A particularly clever solution to this problem, the *weathermap*, has been developed by meteorologists. On one screen, this can convey information such as the general weather patterns as well as the location of critical weather areas.

By analogy, a grid *weathermap* can be used to tell the user "at a glance" what the overall quality metrics of a grid are and in particular which areas of the grid need improvement. This is done in the present work by the superposition, on the graphics screen, of:

- **node-based grid quality measures** - Certain grid quality measures are node-based, that is, they are defined to be valid at a given node. Examples include the ratio of maximum-to-minimum volumes of elements attached to a given node or the number of edges incident at a node. Specific examples are described below. Nodes whose quality measure fall outside some user-specified tolerance band are displayed as a color-coded symbol (the color indicates which quality measure is out of bounds). In addition, it has been found to be useful to connect neighboring nodes which exceed the threshold with the appropriate grid lines.
- **cell-based grid quality measures** - Other grid quality measure are valid for a grid cell. Examples include the skew of a cell; again specific examples are described below. The cells whose grid quality measures exceed the user-specified thresholds are drawn directly. Using this technique, not only are "out of tolerance" areas of the grid highlighted, but by showing the shape of the offending cell(s), the user is better able to determine the cause of the problem.
- **the configuration** - This gives the user a sense of context, that is where the problems are in relation to each other and to the critical parts of the given geometry. This too aids the user in isolating the cause of the problem.

It is a specific strength of the design of the *weathermap* technique to allow users to both adjust the tolerance of the given measures as well as add any new measures which are appropriate to a specific field solver.

Grid Browsing and Querying—Once an area has been identified as being critical, methods are needed to view and analyze the behavior of the cell and the nodes in that region. This gives the user a method for understanding the problem and hence its cause. Many techniques have been tried, but most are confusing.

Current techniques generally do not give the user a direct three-dimensional view of the data, but rather require the user to mentally assemble a three-dimensional view from a series of two-dimensional slices.

The visualization tool which was developed here to browse block-structured grids gives the user an actual three-dimensional view of an i, j, or k-plane of grid points by constructing a three-dimensional model of the grid cells adjacent to the selected plane. This is accomplished by superimposing opaque, lighted representations of the bottoms and sides of the appropriate cells, thus creating an image which looks similar to an *egg carton*. By viewing into the cells, the user can get a realistic view of the shape of the cell and its size relative to the neighbor. With the addition of motion parallax (via interactive zooming, panning, and rotating), the three-dimensional nature of the cells is even more apparent.

An additional difference between the current and traditional techniques is that the new technique automatically extends the grid plane in the current block logically into its neighboring blocks. In this way, the user gets a direct sense of the entire grid, that is not a block-by-block snapshot. Of course, sometimes the user wants to examine a block which has been graphically disconnected from its neighbors. In order to accomplish this, the user has the ability of exploding the grid; this is analogous to the exploded (or assembly) view which is often found in service manuals.

A similar visualization technique is applicable to unstructured grids. The difficulties here are the lack of grid planes; so instead, one must substitute a plane in physical space to select the cells to display. Also, determining which face to not show so that a user can look into the cell is not as easy as it was in structured grids. For these reasons, the *egg carton* has not yet been implemented for unstructured grids.

For both block-structured and unstructured grids, the user can query the system for information through a simple graphical pick. The information returned to the user consists of the identity and attributes (location and quality metrics) of the picked entity, as well as the identity and attributes of all entities connected to it. This operation can be applied recursively to form progressively larger shells. The numerical information returned by the querying mechanism has proven to be extremely useful in debugging some of the grids which have been generated to date.

## SUMMARY

The various components of the NGP system have been presented and can be summarized as follows:

- The **user interface** is designed to be easily customized, extended and ported to a variety of workstation hardware platforms. The entire interface environment can be modified by the user either interactively through the interface, or through "resource" files. New applications can be added with minimum effort. X-windows/Motif is used to enhance portability due to its availability on several types of hardware platforms. The same global metaphors are used for user interaction throughout NGP to ensure consistency between applications.
- The **data structure** is based on a Boundary Representation (B-Rep) radial edge non-manifold solid modeling data structure for both surface and grid topology. All geometry and grid connectivities are explicitly defined. The user is abstracted from the underlying geometry orientation, and a set of algorithms is provided that simplifies grid generation, surface interrogation and geometry construction.
- The **Computer Aided Design (CAD)** system uses a Non-Uniform Rational B-Spline (NURBS) geometry representation. Geometry can be imported into the NGP via the Initial Graphics Exchange Specification (IGES), discrete XYZ's or can be created by the internal CAD system. Geometry read into the NGP system is converted to a NURBS representation. CAD tools are available within the

system that allow the user to build or modify points, curves and surfaces. An important feature of the CAD system is the ability to lay a "carpet" surface over several surfaces to fix defective geometry definitions having gaps and overlaps between surfaces.

- The **automatic blocking system** provides a suite of tools for quickly generating the set of edges, faces, and blocks which define the framework for a block structured grid. This block structure is automatically generated given just an abstraction of the configuration, simple blocking commands (such as wrap a grid around the wing leading edge), and other descriptive information (such as the number of points in various regions and any reguired clusterings). Block structures generated with this technique generally require about an order of magnitude less labor to generate than is required by the traditional constructive techniques used in other block structured grid generators.

- The **structured grid generation** system enables the user to create both 2D and 3D structured grids. Surface grids are calculated in the parametric space of the underlying NURBS. Both surface and volume grid generation are available for 3D, and planar grid generation is available for 2D. Automated face and block detection algorithms allow the user to concentrate on building the blocking structure around a complex geometry without the burden of having to define face-to-face and block-to-block orientations. Elliptic smoothing across domains is available on the surfaces and in the volume.

- The **unstructured grid generation** system enables the user to create both 2D and 3D unstructured grids using both Delauney and Advancing front methods. All surface grids are created in parametric space on the NURBS. 2D grid generation uses wire frame geometry while 3D grid generation uses surface geometry. Automated loop detection is available for 2D grid generation and surface trimming. Surface and volume grid smoothing and optimization options are also available.

- The **visualization** system provides diagnostic feedback on the quality of the grids and selected CFS solutions calculated on the grids. A number of viewing options are available for both structured and unstructured grids. Plane sweeping through multiple blocks, *egg-carton* and *weathermap* options are available for structured grids. *Point cloud* and *weathermap* options are available for unstructured grids.

## REFERENCES


1. Thompson, J.F.: The National Grid Project, Computing Systems in Engineering, Vol. 3, Nos 1-4, pp. 393-399, 1992.

2. OpenGL Architecture Review Board, *OpenGL$^{TM}$ Reference Manual: The Official Reference Document for OpenGL, Release 1*, Addision-Wesley Publishing Company, 1992.

3. Gaither, A.: A Topology Model for Numerical Grid Generation. *Numerical Grid Generation in Computational Fluid Dynamics and Related Fields*, Proceedings of the 4th International Conference, Pineridge Press Ltd, Swansea, U.K. April 1994.

4. Weiler, K.J.: Topological Structures for Geometric Modeling, PHD Dissertation, Rensselaer Polytechnic Institute, Troy, New York, 1986.

5. Farin, G.: Curves and Surfaces for Computer Aided Geometric Design. Third Edition, Academic Press, San Diego, CA, 1992.

6. Mortensen, M.E.: Geometric Modeling. Wiley & Sons, New York, NY, 1985.

7. Piegl, L.: On NURBS: A Survey. IEEE Computer Graphics & Applications, pp. 55. January 1991.

8. Yamaguchi, F.: Curves and Surfaces in Computer Aided Geometric Design. Springer-Verlag, New York, NY, 1988.

9. Hoschek, J.; and Lasser, D.: Fundamentals of Computer Aided Geometric Design. A K Peters, Ltd., Wellesley, MA, 1993.

10. *The Initial Graphics Exchange Specifications (IGES) Version 5.1*, National Computer Graphics Association, IGES/PDES Organization, 1991.

11. Blake, M.W.; Chou, J.J.; Kerr, P.A.; and Thorp, S.A.: The NASA-IGES Geometry Data Exchange Standard. NASA CP-3143, April 1992.

12. Steinbrenner, J.P.; Chawner, J.R.; and Fouts, C.L.: The GRIDGEN 3D Multiple Block Grid Generation System, WRDC-TR-90-3022, Vol. I & II. 1990.

13. Walatka, W.Z.; Buning, P.G.; Pierce, L.; and Elson, P.A.: PLOT3D User's Manual, Version 3.6, NASA TM-101067, 1990.

14. Akdag, V.; and Wulf, A.: Integrated Geometry and Grid Generation System for Complex Configurations. NASA CP-3143, April 1992.

15. Bertin, D.;, et al: A New Automatic Grid Generation Environment for CFD Applications, *Numerical Grid Generation in Computational Fluid Dynamics and Related Fields*, Proceedings of the 4th International Conference, Pineridge Press Ltd, Swansea, U.K. April 1994.

16. Khamayseh, A.: Elliptic Surface Grid Generation on Analytically Defined Geometries. PHD thesis, Mississippi State University, Mississippi State, MS, 1994.

17. Jean, B. A.; and Hamann, B.: Interactive Techniques for Correcting CAD data, *Proceedings of the $\frac{1}{4}^{th}$ International Conference on Numerical Grid Generation in Computational Fluid Dynamics and Related Fields*, April 1994.

18. Dannenhoffer, J.: Automatic Blocking for Complex 3D Configurations. NASA CP-3291, May 1995.

19. Chan, W.M.; Chiu, I.; and Buning, P.G.: User's Manual for The HYPGEN Hyperbolic Grid Generator and the HGUI Graphical User Interface, NASA TM-108791, 1993.

20. Chan, W. M.; and Steger, J. L.: Enhancements of a Three-Dimensional Hyperbolic Grid Generation Scheme. Elseveir Science Publishing Co., New York, NY, 1992.

21. Weatherill, N.P.: Adaptive Inviscid Flow Solutions for Aerospace Geometries on Efficiently Generated Unstructured Tetrahedral Meshes, AIAA-93-0341, 1993.

22. Marcum, D.L.; and Weatherill, N.P.: Unstructured Grid Generation Using Iterative Point Insertion and Local Reconnection. AIAA-94-1926, June 1994.

23. Marcum, D.L.: Generation of Unstructured Grids for Viscous Flow Applications. AIAA-95-0212, January 1995.

24. Remotigue, M.G.; Hart, E.T.; and Stokes, M.L.: EAGLEView: A Surface and Grid Generation Program

and Its Data Management. NASA CP-3143, April 1992.

25. Sorenson, R. L.; and Mccann K. M.: GRAPEVINE: Grids about Anything Poisson's Equation in a Visually Interactive Networking Environment. NASA CP-3143, April 1992.

26. Parmley, K.: Techniques for the Visual Evaluation of Computational Grids. AIAA-93-3353, July 1993.

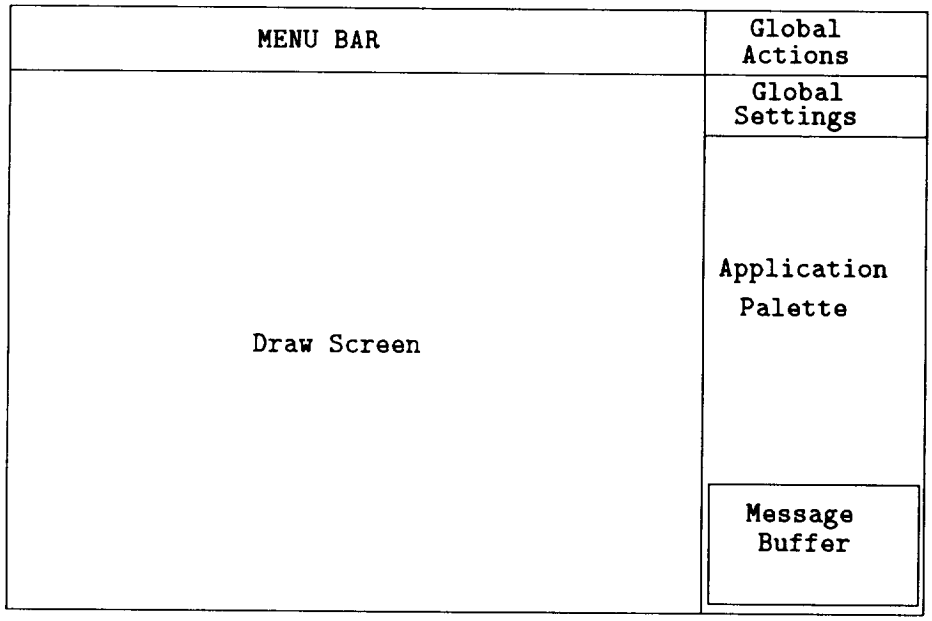



Figure 1: GUI layout of NGP

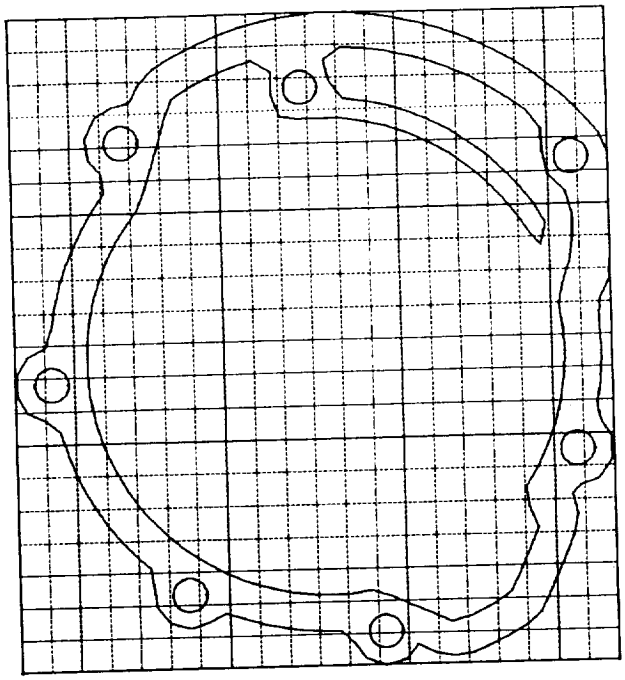


Figure 2: Untrimmed geometry of gasket.

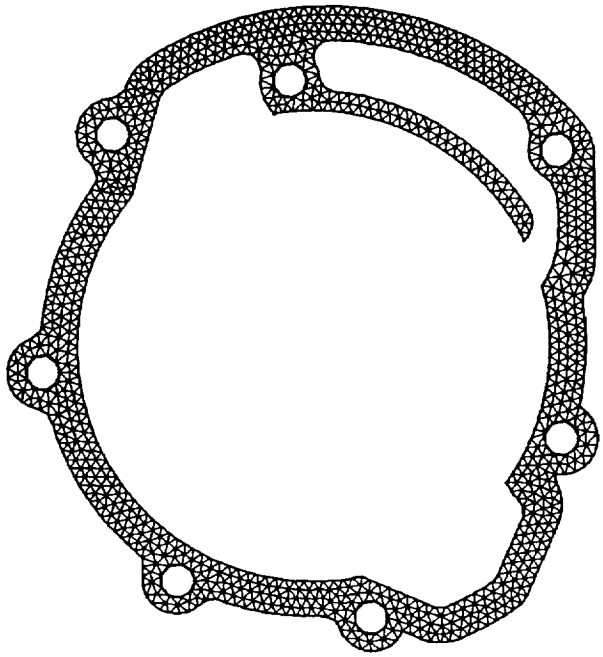


Figure 3: Trimmed geometry of gasket.

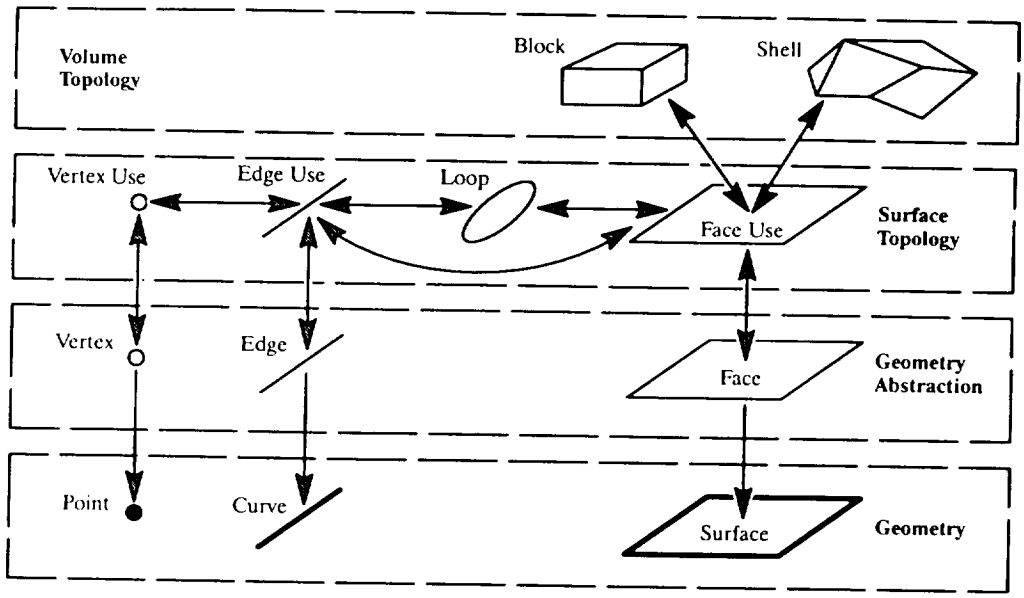



Figure 4: Topology hierarchy.

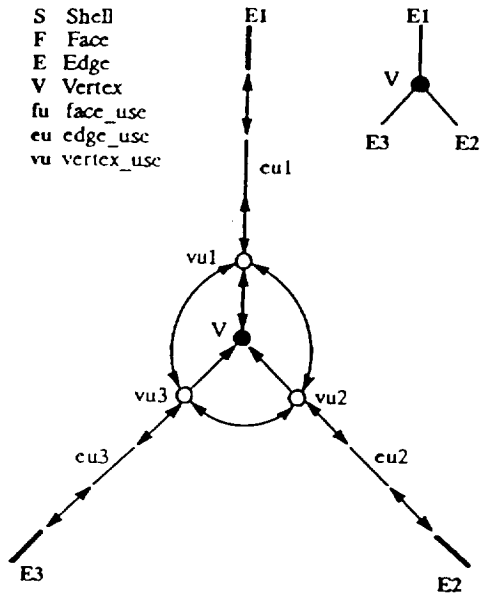



Figure 5: Topology of three edges sharing a vertex.

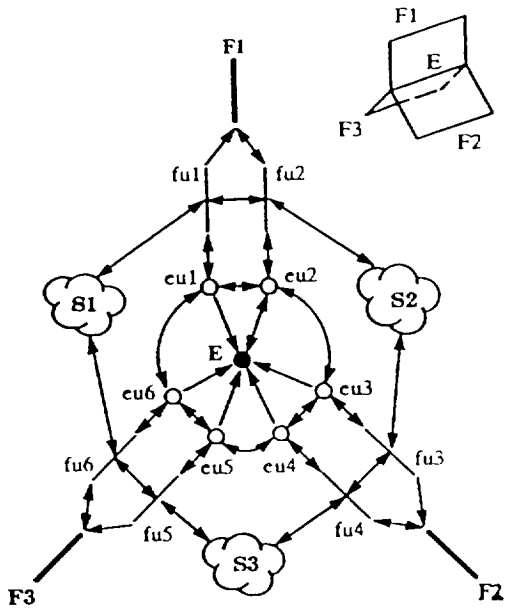



Figure 6: Topology of three faces sharing an edge.

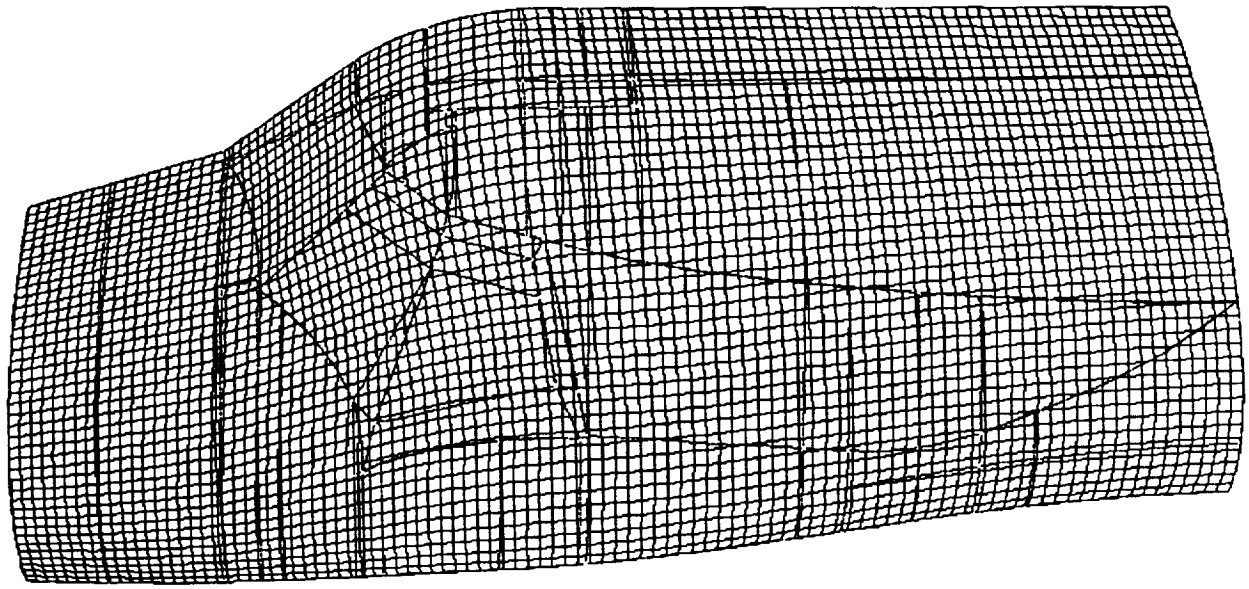


Figure 7: 101x41 carpet grid over network of shuttle surfaces.

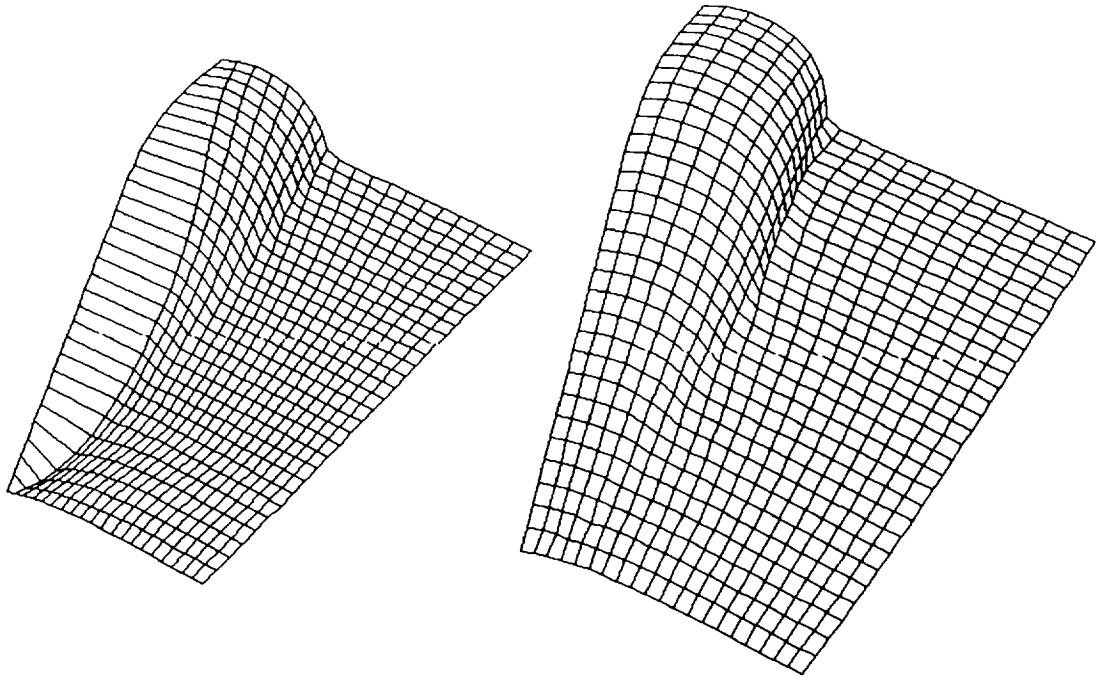


Figure 8: Original grid and algebraic adaption on canopy/fuselage configuration

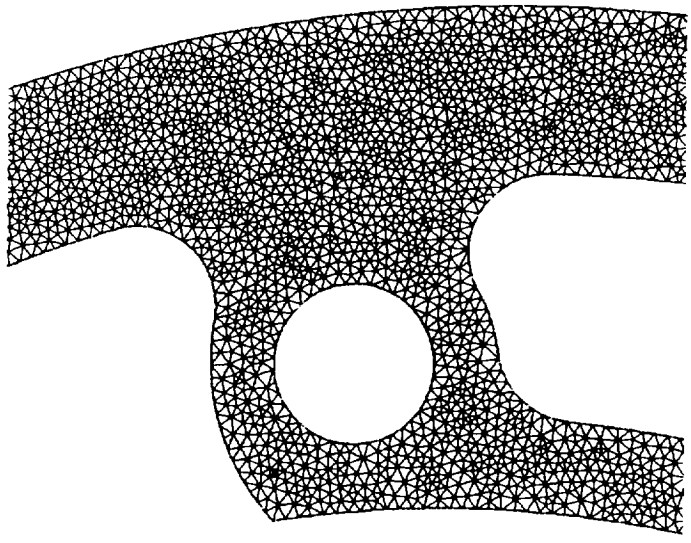


Figure 9: Delaunay unstructured grid on gasket.

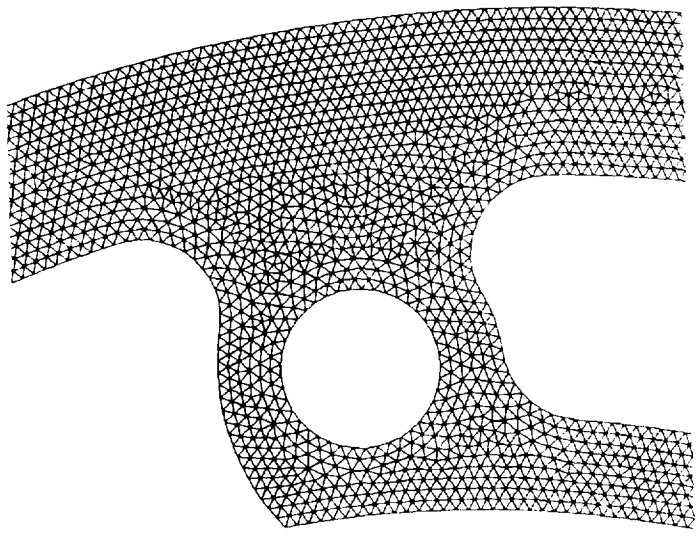


Figure 10: Advancing front unstructured grid on gasket.

Figure 11: Unstructured pointcloud on inlet configuration.

# 3DGRAPE/AL: THE AMES / LANGLEY TECHNOLOGY UPGRADE

Reese L. Sorenson
National Aeronautics and Space Administration
Ames Research Center
Moffett Field, CA 94035-1000

Stephen J. Alter
Lockheed Engineering & Sciences Co.
Langley Research Center
Hampton, VA 23666

## SUMMARY

This paper describes a new three-dimensional structured multiple-block volume grid generator called 3DGRAPE/AL. It is a significantly improved version of the previously-released and widely-distributed program 3DGRAPE, with many of the improvements taken from the grid-generator program 3DMAGGS[1]. It generates volume grids by iteratively solving the Poisson Equations in three-dimensions. The right-hand-side terms are designed so that user-specified grid cell heights and user-specified grid cell skewness near boundary surfaces result automatically, with little user intervention. Versatility was a high priority in this code's development, and as a result it can generate grids in almost any three-dimensional physical domain. Improvements include added kinds of forcing functions, improved control of cell skewness, improved initial conditions, convergence acceleration, the ability to take as input the output from GRIDGEN, and a simple but powerful graphical user interface(GUI).

## INTRODUCTION

The original program, 3DGRAPE[2,3], of which 3DGRAPE/AL is an updated version, is a batch-type program. It reads in pre-defined input data, generates the grid, and writes it out. For those boundary surfaces which are of interest (i.e., the body) it expects to read X,Y,Z coordinates of surface grid points which the user has pre-defined using other software. Other boundary surfaces of less interest (i.e., the outer boundary) can be found by the program itself using simple analytic shapes. The grid can consist of multiple blocks, and the program is capable of finding its own internal block-to-block boundary surfaces. Volume grid points are found by numerically solving the Poisson equations. The Steger & Sorenson (S&S) Right-Hand-Side (RHS) terms (i.e., forcing functions) in those equations are of a type which allows the user to choose the desired cell height on a read-in boundary, after which the program automatically finds the actual numerical values for the RHS terms which yield the desired cell heights. In the process the RHS terms attempt to give local near-orthogonality in the region of those same read-in surfaces. The cell heights the user requires may be of any magnitude (limited only by the precision of the computer), appropriate for both viscous and inviscid aerodynamic flow modeling. The input data is ordinary text, with required formatting. The output grid may be any of three formats, including the commonly used PLOT3D[4] formats.

All the features described above for the original program are preserved in the new program, and a significant suite of new features is added. Those new features include:

- Grid quality is enhanced by re-formulated Steger & Sorenson (S&S) control terms in the Poisson Equations. The user may specify arbitrary angles with which lines are to intersect boundaries, rather than that specification being limited to 90° everywhere. The treatment of sharp corners which transverse boundary surfaces (e.g., a grid wrapping around an airplane fuselage which has a strake) is improved using this capability.
- Another improvement to grid quality is the addition of Thomas & Middlecoff (T&M) clustering terms for cases where all six faces of a block are read-in, as found in GRIDGEN and 3DMAGGS. The user can choose either the Steger & Sorenson terms (as in the original code and improved as described above), the

Thomas & Middlecoff type terms, or a blending between the two which gives good cell-size and skewness control at both the boundaries and the interior.

- Grid quality is evaluated by computing and printing maxima, minima, medians, and averages of cell heights and non-orthogonality, at boundaries and in the interiors of the blocks of the finished grid.
- Initialization is improved by Trans-Finite Interpolation[5] (for cases with six fixed boundary surfaces). In some cases grids initialized thusly can serve as the final grid, in others this improved initialization speeds convergence.
- Erlich's Ad Hoc Method for computing locally optimum relaxation parameters is available for the code's SOR solver. This also can accelerate convergence
- When installed on CRAY computers the code is vectorized in all three coordinate directions, allowing the longest possible vector length in each block. This, too, accelerates convergence.
- The grid generation iteration schedule can be divided into parts. Parameters which effect convergence (such as relaxation rates), as well as the type of clustering terms used and their associated decay rates, are adjustable with each part. Intermediate solutions and restart files can be written after each part. Thus, in practical operation, as much can sometimes be accomplished in one run with this program as in multiple runs with other grid generators.
- An input filter called PREGRAPE/AL, taken from 3DMAGGS, is supplied as a companion program. It inputs the output from the GRIDGEN[6] code, which contains blocking strategy and surface grids, and turns that into input for 3DGRAPE/AL.
- Required cell heights and skewness at read-in surfaces can be specified by the user at each point from a file. One possible application of the ability to specify the skewness at each point is in hypersonic flow where in the exit plane near the shock the flow is aligned with the shock and so should be the grid, while in that same face near the body the grid should be aligned with the body surface, and the angles are blended in-between.
- A complete grid generated elsewhere can be read-in, and the elliptic solver can be run a few steps to smooth the grid.
- A Graphical User Interface, coded in FORTRAN-77 and calling the IRIS Graphics Library, allows the user to watch selected grid surfaces while the grid solver is iterating. A full suite of transforms and other features is included.

The code is written in FORTRAN-77. It can be installed as an ordinary batch program, and in that form it should run on almost any computer. Alternately, on a Silicon Graphics Inc. (SGI) workstation it can be installed along with its graphical user interface. The GUI is also written in FORTRAN-77, and calls functions in the IRIS Graphics Library. For compiling on a CRAY supercomputer there is a vectorized batch version.

## THEORETICAL DEVELOPMENT OF THE POISSON EQUATIONS IN PHYSICAL SPACE

The original 3DGRAPE program, the 3DMAGGS program, and the new 3DGRAPE/AL program all generate grids by iteratively solving the Poisson Equations in three-dimensions. A mapping is thus found between the computational coordinates $\xi,\eta,\zeta$ and the physical coordinates X,Y,Z. The equations are typically given in the computational space as

$$\xi_{xx} + \xi_{yy} + \xi_{zz} = P(\xi,\eta,\zeta) \tag{1a}$$

$$\eta_{xx} + \eta_{yy} + \eta_{zz} = Q(\xi,\eta,\zeta) \tag{1b}$$

$$\zeta_{xx} + \zeta_{yy} + \zeta_{zz} = R(\xi,\eta,\zeta) \tag{1c}$$

However, it is natural to apply them in the physical space. It is natural to specify the grid boundary conditions by giving X,Y,Z at fixed values of $\xi,\eta,\zeta$ rather than to give values of $\xi,\eta,\zeta$ at fixed values of X,Y,Z. The transformation of Eqs. 1 to physical space proceeds as follows. Clearly, we must have

$$\xi = \xi(x,y,z) \tag{2a}$$

$$\eta = \eta(x,y,z) \tag{2b}$$

$$\zeta = \zeta(x,y,z) \tag{2c}$$

To effect this transformation we must also have

$$x = x(\xi,\eta,\zeta) \tag{3a}$$

$$y = y(\xi,\eta,\zeta) \tag{3b}$$

$$z = z(\xi,\eta,\zeta) \tag{3c}$$

Differentiating Eqs. 2 and applying the chain rule gives

$$\begin{vmatrix} d\xi \\ d\eta \\ d\zeta \end{vmatrix} = \begin{vmatrix} \xi_x & \xi_y & \xi_z \\ \eta_x & \eta_y & \eta_z \\ \zeta_x & \zeta_y & \zeta_z \end{vmatrix} \begin{vmatrix} dx \\ dy \\ dz \end{vmatrix} \tag{4}$$

Likewise, differentiating Eqs. 3 and applying the chain rule gives

$$\begin{vmatrix} dx \\ dy \\ dz \end{vmatrix} = \begin{vmatrix} x_\xi & x_\eta & x_\zeta \\ y_\xi & y_\eta & y_\zeta \\ z_\xi & z_\eta & z_\zeta \end{vmatrix} \begin{vmatrix} d\xi \\ d\eta \\ d\zeta \end{vmatrix} \tag{5}$$

We designate the 3 x 3 matrix in Eq. 5 as M, assume that its inverse exists, and pre-multiply both sides of Eq. 5 by $M^{-1}$. This gives

$$M^{-1} \begin{vmatrix} dx \\ dy \\ dz \end{vmatrix} = \begin{vmatrix} d\xi \\ d\eta \\ d\zeta \end{vmatrix} \tag{6}$$

Substituting from Eq. 6 into Eq. 4 gives

$$M^{-1}\begin{vmatrix} dx \\ dy \\ dz \end{vmatrix} = \begin{vmatrix} \xi_x & \xi_y & \xi_z \\ \eta_x & \eta_y & \eta_z \\ \zeta_x & \zeta_y & \zeta_z \end{vmatrix} \begin{vmatrix} dx \\ dy \\ dz \end{vmatrix} \tag{7}$$

We know that, in general, if

$$A\vec{v} = B\vec{v} \tag{8a}$$

and if $B^{-1}$ exists then

$$B^{-1} A \vec{v} = \vec{v} \tag{8b}$$

Therefore it must be true that

$$B^{-1} A = I \tag{8c}$$

Pre-multiplying by B gives

$$A = B \tag{8d}$$

Applying Eqs. 8 to Eq. 7 gives

$$M^{-1} = \begin{vmatrix} \xi_x & \xi_y & \xi_z \\ \eta_x & \eta_y & \eta_z \\ \zeta_x & \zeta_y & \zeta_z \end{vmatrix} \tag{9}$$

For this to be useful, we must find $M^{-1}$. It is known, in general, that

$$A^{-1} = \frac{Adj(A)}{Det(A)} \tag{10}$$

Where Adj(A) is the adjoint of A and Det(A) is the determinant of A. The adjoint of A is a matrix having as each element the corresponding cofactor of A. Thus, from Eq. 9, we have

$$\begin{vmatrix} \xi_x & \xi_y & \xi_z \\ \eta_x & \eta_y & \eta_z \\ \zeta_x & \zeta_y & \zeta_z \end{vmatrix} = \begin{vmatrix} \gamma_{11} & \gamma_{12} & \gamma_{13} \\ \gamma_{21} & \gamma_{22} & \gamma_{23} \\ \gamma_{31} & \gamma_{32} & \gamma_{33} \end{vmatrix} / J \tag{11}$$

where $\gamma_{ij}$ is the ij-th cofactor of M and J is the determinant of M. By inspection of Eq. 11 we see that

$$\xi_x = \gamma_{11}/J \tag{12a}$$

$$\xi_y = \gamma_{12}/J \tag{12b}$$

$$\xi_z = \gamma_{13}/J \tag{12c}$$

$$\eta_x = \gamma_{21}/J \tag{12d}$$

$$\eta_y = \gamma_{22}/J \tag{12e}$$

$$\eta_z = \gamma_{23}/J \tag{12f}$$

$$\zeta_x = \gamma_{31}/J \tag{12g}$$

$$\zeta_y = \gamma_{32}/J \tag{12h}$$

$$\zeta_z = \gamma_{33}/J \tag{12i}$$

Completion of the derivation of the transformed Poisson equations requires further differentiating the metrics in Eqs. 12, substituting them into Eqs. 1, and collecting terms. This process is simple calculus, but very lengthy and beyond the scope of this paper. The result is

$$\begin{aligned} &\alpha_{11}\vec{r}_{\xi\xi} + \alpha_{22}\vec{r}_{\eta\eta} + \alpha_{33}\vec{r}_{\zeta\zeta} \\ &+ 2\,(\,\alpha_{12}\vec{r}_{\xi\eta} + \alpha_{13}\vec{r}_{\xi\zeta} + \alpha_{23}\vec{r}_{\eta\zeta}) \\ &= -J^2\,(P\vec{r}_{\xi} + Q\vec{r}_{\eta} + R\vec{r}_{\zeta}) \end{aligned} \tag{13a}$$

where:

$$\vec{r} = \begin{vmatrix} \overrightarrow{x} \\ y \\ z \end{vmatrix} \tag{13b}$$

and

$$\alpha_{ij} = \sum_{m=1}^{3} \gamma_{mi}\gamma_{mj} \tag{13c}$$

## THEORETICAL DEVELOPMENT OF IMPROVED S&S RHS TERMS

The distribution of points in the grid results primarily from the influence of the Right-Hand Side (RHS) terms, or forcing functions. We are free to choose them as we please. In both new and old programs they are:

$$\begin{aligned} P(\xi,\eta,\zeta) = {} & P_1(\eta,\zeta)e^{-a\xi} + P_2(\eta,\zeta)e^{-a(\xi_{max}-\xi)} \\ + {} & P_3(\xi,\zeta)e^{-a\eta} + P_4(\xi,\zeta)e^{-a(\eta_{max}-\eta)} \\ + {} & P_5(\xi,\eta)e^{-a\zeta} + P_6(\xi,\eta)e^{-a(\zeta_{max}-\zeta)} \end{aligned} \tag{14a}$$

$$Q(\xi,\eta,\zeta) = Q_1(\eta,\zeta)e^{-a\xi} + Q_2(\eta,\zeta)e^{-a(\xi_{max}-\xi)}$$
$$+ \; Q_3(\xi,\zeta)e^{-a\eta} + Q_4(\xi,\zeta)e^{-a(\eta_{max}-\eta)} \qquad (14b)$$
$$+ \; Q_5(\xi,\eta)e^{-a\zeta} + Q_6(\xi,\eta)e^{-a(\zeta_{max}-\zeta)}$$

$$R(\xi,\eta,\zeta) = R_1(\eta,\zeta)e^{-a\xi} + R_2(\eta,\zeta)e^{-a(\xi_{max}-\xi)}$$
$$+ \; R_3(\xi,\zeta)e^{-a\eta} + R_4(\xi,\zeta)e^{-a(\eta_{max}-\eta)} \qquad (14c)$$
$$+ \; R_5(\xi,\eta)e^{-a\zeta} + R_6(\xi,\eta)e^{-a(\zeta_{max}-\zeta)}$$

Clearly, these RHS terms P,Q,R are simply superpositions of other terms $P_n$,$Q_n$,$R_n$ for $1 \le n \le 6$, multiplied by exponentials which are at their maximum value, one, at the boundary surfaces and which decay with distance into the interior of the block. The positive constant "a" in Eqs. 14 is set by the user, and determines the rate of exponential decay in the size and influence of the RHS terms.

At this point we must introduce a nomenclature for the face numbers. It is seen in Table I. By examining that nomenclature we see that at each of the boundaries the terms in P,Q,R having their subscripts equal to the face number are non-zero, and the other terms in P,Q,R approach zero due to the behavior of their exponential factors. At face 3, for example, Eqs. 14 reduce to:

$$P(\xi,\eta,\zeta) = P_3(\xi,\zeta) \qquad (15a)$$

$$Q(\xi,\eta,\zeta) = Q_3(\xi,\zeta) \qquad (15b)$$

$$R(\xi,\eta,\zeta) = R_3(\xi,\zeta) \qquad (15c)$$

So then we can find the terms $P_n$,$Q_n$,$R_n$ at face n by considering each face in turn. At each point on each face we:

- Assume that the Poisson Equations, Eqs. 13, are satisfied.
- Find values for all first and second partial derivatives required by Eqs. 13.
- Eqs. 13 reduce to a 3 x 3 set of linear equations in the three unknowns $P_n$,$Q_n$,$R_n$. Solve them.

Having found all the $P_n$,$Q_n$,$R_n$, for $1 \le n \le 6$, we can calculate P,Q,R at all points in the grid from Eqs. 14.

However, finding values for all first and second partial derivatives at each face is not trivial. To further illustrate this we must restrict our attention to a particular face. We choose face 3 to illustrate. On face 3 the derivatives $\vec{r}_\xi$, $\vec{r}_\zeta$, $\vec{r}_{\xi\xi}$, $\vec{r}_{\xi\zeta}$, and $\vec{r}_{\zeta\zeta}$ can be found by differencing known boundary face points. The derivatives $\vec{r}_{\eta\eta}$ are found by differencing the grid solution at the current time step, as described on page 78 of Ref. 2. If we could find derivatives $\vec{r}_\eta$ we could then difference them to find derivatives $\vec{r}_{\xi\eta}$ and $\vec{r}_{\eta\zeta}$.

We find derivatives $\vec{r}_\eta$ by adding additional equations which embody the user's requirements on cell height and skewness. In the old 3DGRAPE method we added the three equations

$$\vec{r}_\xi \cdot \vec{r}_\eta = 0 \qquad (16a)$$

$$\vec{r}_\eta \cdot \vec{r}_\zeta = 0 \qquad (16b)$$

$$\vec{r}_\eta \cdot \vec{r}_\eta = S^2 \tag{16c}$$

As seen in Table 1, $\xi$ and $\zeta$ vary over face 3, and $\eta$ varies along lines intersecting the face. Thus Eqs. 16a and 16b require orthogonality between the lines intersecting the face and the coordinate lines running over the face. Eq. 16c requires that the cell height on the surface be the positive constant S.

It is at this point that the old 3DGRAPE method and the new 3DGRAPE/AL method differ. In the new method we realize that when making grids about real-world configurations, with singularities and slope discontinuities, it is sometimes necessary to have grid cells which are skewed in a specified way. Lacking this ability, an inconsistency can develop which can either cause the elliptic solver to not converge, or result in an unsuitable grid. And so Eqs. 16 are replaced by

$$\vec{r}_\xi \cdot \vec{r}_\eta = |\vec{r}_\xi| \cdot |\vec{r}_\eta| \cos\theta_1 \tag{17a}$$

$$\vec{r}_\eta \cdot \vec{r}_\zeta = |\vec{r}_\eta| \cdot |\vec{r}_\zeta| \cos\theta_2 \tag{17b}$$

$$\vec{r}_\eta \cdot \vec{r}_\eta = S^2 \tag{17c}$$

where $\theta_1$ is the angle between the coordinate line intersecting face 3 and the line of varying $\xi$ on face 3, and $\theta_2$ is the angle between the coordinate line intersecting face 3 and the line of varying $\zeta$ on face 3. For $\theta_1$ and $\theta_2$ equal to 90°, Eqs. 17 reduce to Eqs. 16.

We now proceed to solve Eqs. 17 for $\vec{r}_\eta$. Expanding, we have

$$x_\xi x_\eta + y_\xi y_\eta + z_\xi z_\eta = c_1 \tag{18a}$$

$$x_\zeta x_\eta + y_\zeta y_\eta + z_\zeta z_\eta = c_2 \tag{18b}$$

$$x_\eta^2 + y_\eta^2 + z_\eta^2 = S^2 \tag{18c}$$

where

$$c_1 = |\vec{r}_\xi| \, S \cos\theta_1$$

$$c_2 = |\vec{r}_\zeta| \, S \cos\theta_2$$

$c_1$ and $c_2$ are constants because $\theta_1$, $\theta_2$, S, and the points on face 3 are user-defined inputs. Equations 18 are three equations in the three unknowns $x_\eta, y_\eta, z_\eta$ which are the elements of $\vec{r}_\eta$. But because Eq. 18c is quadratic, solving this set of equations is not straightforward. We will make an assumption about one of the unknowns and solve, make that assumption about another of the unknowns and solve, and then make that assumption about the last of the unknowns and solve. We will then select the answer which is "best."

The first assumption we make is that $x_\eta$ is a constant. Terms involving $x_\eta$ in Eqs. 18a and 18b are brought to the right side of the equations, and then the equations are solved, yielding

$$y_\eta = x_\eta \gamma_{22} / \gamma_{12} + k_1 \tag{19a}$$

$$z_\eta = x_\eta \gamma_{32} / \gamma_{12} + k_2 \tag{19b}$$

where

$$k_1 = \frac{c_1 z_\zeta - c_2 z_\xi}{-\gamma_{12}}$$

$$k_2 = \frac{c_2 y_\xi - c_1 y_\zeta}{-\gamma_{12}}$$

K1 and k2 are constants. Then from Eq. 18c

$$x_\eta = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a} \tag{20}$$

where

$$a = 1 + (\gamma_{22}/\gamma_{12})^2 + (\gamma_{32}/\gamma_{12})^2$$

$$b = \frac{2}{\gamma_{12}}(k_1\gamma_{12} + k_2\gamma_{32})$$

$$c = k_1^2 + k_2^2 - S^2$$

The second assumption is that $y_\eta$ is a constant. Terms involving $y_\eta$ in Eqs. 18a and 18b are brought to the right side of the equations, and then the equations are solved, yielding

$$x_\eta = y_\eta\gamma_{12} / \gamma_{22} + k_1 \tag{21a}$$

$$z_\eta = y_\eta\gamma_{32} / \gamma_{22} + k_2 \tag{21b}$$

where

$$k_1 = \frac{c_1 z_\zeta - c_2 z_\xi}{\gamma_{22}}$$

$$k_2 = \frac{c_2 x_\xi - c_1 x_\zeta}{\gamma_{22}}$$

Then from Eq. 18c

$$y_\eta = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a} \tag{22}$$

where

$$a = 1 + (\gamma_{12}/\gamma_{22})^2 + (\gamma_{32}/\gamma_{22})^2$$

$$b = \frac{2}{\gamma_{22}}(k_1\gamma_{12} + k_2\gamma_{32})$$

and c is the same as above, in Eq. 20.

The third assumption is that $z_\eta$ is a constant. Terms involving $z_\eta$ in Eqs. 18a and 18b are brought to the right side of the equations, and then the equations are solved, yielding

$$x_\eta = z_\eta \gamma_{12} / \gamma_{32} + k_1 \tag{23a}$$

$$y_\eta = z_\eta \gamma_{22} / \gamma_{32} + k_2 \tag{23b}$$

where

$$k_1 = \frac{c_1 y_\zeta - c_2 y_\xi}{-\gamma_{32}}$$

$$k_2 = \frac{c_2 x_\xi - c_1 x_\zeta}{-\gamma_{32}}$$

Then from Eq. 17c

$$z_\eta = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a} \tag{24}$$

where

$$a = 1 + (\gamma_{12}/\gamma_{32})^2 + (\gamma_{22}/\gamma_{32})^2$$

$$b = \frac{2}{\gamma_{32}}(k_1 \gamma_{12} + k_2 \gamma_{22})$$

and c is the same as above, in Eq. 20.

In general, none of these three assumptions is strictly correct. However, it usually turns out that at least one of them is close enough to correct for this method to generate suitable grids. It was said that we would choose whichever of these three solutions was "best." However, Eqs. 20, 22, and 24 each include an ambiguous sign from a square-root operation. Therefore, we actually have six solutions to choose from. Using each of the six solutions we compute the Jacobian. If the coordinates in the block are right-handed (with the "handedness" being a user-defined input) we choose the solution which yields the largest positive Jacobian. If the coordinates in the block are left-handed we choose the solution which yields the largest negative Jacobian. The logic behind choosing based upon the Jacobian is that Jacobians, as defined above, having large absolute values seem to be present in grids which are more orthogonal, and, conversely, Jacobians having small absolute values seem to be present in grids which are highly skewed. Thus the elements of $\vec{r}_\eta$ are found.

The foregoing is the analysis for face 3. The analysis for face 4 appears identical, differing only in some of the difference formulas. The analyses for faces 1, 2, 5, and 6 follow in a straightforward manner from the foregoing example.

This formulation for the S&S RHS terms requires a lot of computation but most of it is done only once, at the start of the iteration schedule. It was said, above, that having all values for the derivatives at the face, those derivatives are substituted into Eqs. 13, yielding a 3 x 3 set of linear equations in the three unknowns $P_n, Q_n, R_n$. Their solution shows $P_n, Q_n, R_n$ to be linear functions of the second derivatives $\vec{r}_{\eta\eta}$ which are found by differencing at each time step. The coefficients in those linear functions are fixed for all computational time. Therefore, the only computation necessary to find the RHS terms in each iteration is to re-evaluate $\vec{r}_{\eta\eta}$, re-evaluate the linear functions to get $P_n, Q_n, R_n$ at each face, and then use Eqs. 14 to re-compute the P,Q,R at every point in the grid.

The effectiveness of this method is seen in Figure 1. When wrapping a grid around a sharp edge it is necessary to cause the lines intersecting the surface near the edge to bend toward the edge for best results. The ultimate example of wrapping a grid around a sharp edge is to wrap it around the edge of a flat plate. Figure 1 shows a wing with a zero-thickness extension in the spanwise direction, and a C-H type grid around it. Thus, it is necessary to wrap a C-type grid around the leading-edge of that wing and its flat-plate extension. This would not have been possible with the old type RHS terms.

## THOMAS AND MIDDLECOFF CLUSTERING TERMS

When making grids in regions where all six faces of the computational cube are fixed it is sometimes advantageous to use clustering functions where the spacing normal to a face is determined by the spacing on the side walls. The Thomas and Middlecoff[7] clustering terms are included here for that purpose. However, the Thomas and Middlecoff clustering terms

$$P = \Phi(\nabla\xi \cdot \nabla\xi) \tag{25a}$$

$$Q = \Psi(\nabla\eta \cdot \nabla\eta) \tag{25b}$$

$$R = \Omega(\nabla\zeta \cdot \nabla\zeta) \tag{25c}$$

where

$$\Phi = \frac{\vec{r}_\xi \cdot \vec{r}_{\xi\xi}}{\vec{r}_\xi \cdot \vec{r}_\xi} \tag{25d}$$

$$\Psi = \frac{\vec{r}_\eta \cdot \vec{r}_{\eta\eta}}{\vec{r}_\eta \cdot \vec{r}_\eta} \tag{25e}$$

$$\Omega = \frac{\vec{r}_\zeta \cdot \vec{r}_{\zeta\zeta}}{\vec{r}_\zeta \cdot \vec{r}_\zeta} \tag{25f}$$

are given in the computational space, and to be useful here they must be converted to physical space. Applying the definition of the $\nabla$ operator, illustrated by

$$\nabla\xi = \xi_x\hat{j} + \xi_y\hat{k} + \xi_z\hat{l} \tag{26}$$

where $\hat{j}$, $\hat{k}$, and $\hat{l}$ are the unit normal vectors, and reducing, gives

$$P = \Phi(\xi_x^2 + \xi_y^2 + \xi_z^2) \tag{27a}$$

$$Q = \psi(\eta_x^2 + \eta_y^2 + \eta_z^2) \tag{27b}$$

$$R = \Omega(\zeta_x^2 + \zeta_y^2 + \zeta_z^2) \tag{27c}$$

Substituting the metrics shown in Eqs. 12 into Eqs. 27, and expanding and re-grouping, gives

$$P = \Phi\left[\left(\vec{r}_\eta \cdot \vec{r}_\eta\right)\left(\vec{r}_\zeta \cdot \vec{r}_\zeta\right) - \left(\vec{r}_\eta \cdot \vec{r}_\zeta\right)\right]/J^2 \tag{28a}$$

$$Q = \Psi\left[\left(\vec{r}_\xi \cdot \vec{r}_\xi\right)\left(\vec{r}_\zeta \cdot \vec{r}_\zeta\right) - \left(\vec{r}_\xi \cdot \vec{r}_\zeta\right)\right]/J^2 \tag{28b}$$

$$R = \Omega\left[\left(\vec{r}_\xi \cdot \vec{r}_\xi\right)\left(\vec{r}_\eta \cdot \vec{r}_\eta\right) - \left(\vec{r}_\xi \cdot \vec{r}_\eta\right)\right]/J^2 \tag{28c}$$

These RHS terms generate good grids in many applications. An exception is the situation where the opposing side boundaries, from which the T&M terms are calculated, have very different clustering characteristics. In these cases instabilities in the Poisson solver can result.

It was found in the development of 3DMAGGS that S&S clustering terms tend to give the most-nearly-orthogonal grids near boundaries, while T&M clustering terms give the best clustering in the interior of the blocks. And so a blending between the two kinds of RHS terms was developed, and is included in 3DGRAPE/AL.

## OPTIMUM RELAXATION PARAMETER

3DGRAPE/AL solves the 3-D Poisson equations using Point Successive Over Relaxation (PSOR). In PSOR there is a relaxation parameter, $\Omega$, which determines the rate of convergence and stability of the method. In the old program the $\Omega$ was fixed for all computational time. That option is still available in the new code as well. However, the new code also has an algorithm to compute an optimum relaxation parameter at every point in the grid using the method of Erlich.[8]

That method requires the equations being solved, here Eq. 13a, to be represented as a difference equation of the following form:

$$\begin{aligned} a_0\vec{r}_{j,k,l} + a_1\vec{r}_{j+1,k,l} + a_2\vec{r}_{j,k+1,l} + a_3\vec{r}_{j,k,l+1} \\ + \; a_4\vec{r}_{j-1,k,l} + a_5\vec{r}_{j,k-1,l} + a_6\vec{r}_{j,k,l-1} = \vec{b}_{j,k,l} \end{aligned} \tag{29}$$

Applying standard central differences to all first and second partial derivatives in Eq. 13a, and collecting terms, we arrive at the form of Eq. 29, where

$$a_0 = -2\left(\frac{\alpha_{11}}{(\Delta\xi)^2} + \frac{\alpha_{22}}{(\Delta\eta)^2} + \frac{\alpha_{33}}{(\Delta\zeta)^2}\right) \tag{30a}$$

$$a_1 = \frac{\alpha_{11}}{(\Delta\xi)^2} + \frac{J^2P}{2\Delta\xi} \tag{30b}$$

$$a_2 = \frac{\alpha_{22}}{(\Delta\eta)^2} + \frac{J^2Q}{2\Delta\eta} \tag{30c}$$

$$a_3 = \frac{\alpha_{33}}{(\Delta\zeta)^2} + \frac{J^2R}{2\Delta\zeta} \tag{30d}$$

$$a_4 = \frac{\alpha_{11}}{(\Delta\xi)^2} - \frac{J^2 P}{2\Delta\xi} \tag{30e}$$

$$a_5 = \frac{\alpha_{22}}{(\Delta\eta)^2} - \frac{J^2 Q}{2\Delta\eta} \tag{30f}$$

$$a_6 = \frac{\alpha_{33}}{(\Delta\zeta)^2} - \frac{J^2 R}{2\Delta\zeta} \tag{30g}$$

The complex eigenvalues of Eq. 29 at each point, ignoring wave numbers above 1, are

$$\mu = \mu_r + \mu_i = \frac{2}{a_0}\left(\sqrt{a_1 a_4}\cos\frac{\pi}{j_{max}+1} + \sqrt{a_2 a_5}\cos\frac{\pi}{k_{max}+1} + \sqrt{a_3 a_6}\cos\frac{\pi}{l_{max}+1}\right) \tag{31}$$

where $\mu_r$ and $\mu_i$ are the real and imaginary parts of $\mu$, respectively. It is required that $|\mu_\eta| < 1$.

Continuing with Erlich's method, as formulated by Steinbrenner, Chawner, and Fouts on pages 6-6 and 6-7 of Ref. 6 (with typographical errors corrected), we let

$$A = \mu_r^2 + \mu_i^2 \tag{32a}$$

$$B = \mu_r^2 - \mu_i^2 \tag{32b}$$

$$C = A^2 - B^2 \tag{32c}$$

$$D = A^2 - B \tag{32d}$$

$$E = \sqrt{C + D^2} \tag{32e}$$

$$F = \sqrt[3]{C} \tag{32f}$$

Then

$$\bar{\omega} = \frac{(3D + E)\, F\, \sqrt[3]{E-D} - (3D - E)\, F\, \sqrt[3]{E+D} + A^2 + 3B^2 - 4A^2 B}{A^2 D} \tag{33}$$

and the relaxation parameter $\omega$ is

$$\omega = \begin{cases} -\left(\bar{\omega} - \sqrt{\bar{\omega}^2 + 4\bar{\omega}}\right)/2 & \text{if } D > 0 \\ -\left(\bar{\omega} + \sqrt{\bar{\omega}^2 + 4\bar{\omega}}\right)/2 & \text{if } D < 0 \end{cases} \tag{34}$$

This method can reduce the number of iterations required to achieve convergence. The $\omega$ so computed can sometimes be a little too large, and so cause instabilities. Therefore, in the code, they are multiplied by a limiting factor. The default value of this factor is 0.7, but a value of 0.6 was found to be necessary in one of the

sample cases. These $\omega$ are dependent on the grid at its current time step, and so they are re-calculated each time step. As can be inferred from the above, computing them requires a significant amount of computer time, so the code has an option wherein they are re-calculated every n time steps.

## CURRENT AND FUTURE WORK

Extensions and improvements to 3DGRAPE/AL planned for the near future, or currently in progress, include:

- Adding cylindrical topology -- solving in $\rho,\theta,x$ -- to improve solutions about cylindrical axes, just as use of spherical topology is offered to improve solutions about polar axes.
- Write a translator to translate input for the old program 3DGRAPE into input for the new program 3DGRAPE/AL. This will make datasets used in the old program workable in the new program. Also, the latest release of GRIDGEN, Version 9, can generate input to the earlier 3DGRAPE program, and so this translator will provide an alternate pathway between the two codes.
- We will try again to build multigrid into the Poisson solver. We tried once before, but did not succeed. We know that in grids produced by this code the cell size increases in an approximately exponential fashion with distance from a boundary, but we cannot predict exactly what that rate of increase in cell height will be. Therefore, although we know the desired distance to the 1st node away from the boundary, we cannot say, exactly, what should be the distances to the 2nd, 4th, 8th, 16th, etc., nodes from the boundary. But we need those distances to operate at the various multigrid levels. And so we estimate. But those estimates are not accurate. Thus, there are inconsistencies between the equations being solved at the different multigrid levels, and therefore it does not converge. But we are determined to build in a multigrid capability if at all possible.
- Optimize the code to run faster on SGI machines, such as ONYX. Portability to other platforms will be preserved.
- Write a GUI for PREGRAPE/AL.
- Put in a new boundary condition wherein the angles of lines intersecting a fixed internal boundary is mimicked across that boundary.
- Make a version of the code which uses PVM to run the code in parallel on multiple platforms.

## ACKNOWLEDGMENTS.

The authors are grateful to Professor Joe Thompson for his pioneering work in elliptic grid generation, and for his gracious and encouraging example. Dr. Jeffrey Hultquist of NASA Ames Research Center provided invaluable help with the graphics; were it not for his patience, generosity, and expertise the GUI would not have been possible. Mr. Scott Thomas of Sterling Software assisted in vectorizing the code on the CRAY, and Dr. Jin Chou of Computer Sciences Corporation helped assisted with vector analysis. Dr. Jamshid S. Abolhassani of Computer Sciences Corporation provided various valuable insights and explanations. Mr. William Kleb of NASA Langley Research Center assisted in the formulation of LARCS and other interpolation issues. The second author gratefully acknowledges support from contract NAS1-19000. This work is dedicated to the memory of Joseph L. Steger: scientist, mentor, teacher, and friend.

## REFERENCES


[1]Alter, S. J., Weilmuenster, K. J., "The Three-Dimensional Multi-Block Advanced Grid Generation System (3DMAGGS)," NASA TM 108985, May, 1993.

[2]Sorenson, R. L., "The 3DGRAPE Book: Theory, Users' Manual, Examples," NASA TM 102224, July, 1989.

[3]Sorenson, R. L., "Three-Dimensional Zonal Grids About Arbitrary Shapes by Poisson's Equation," appearing in Sengupta, S., Häuser, J., Eiseman, P.R., and Taylor, C., eds., Numerical Grid Generation in Computational Fluid Mechanics, Pineridge Press Ltd., 1988.

[4]Walatka, P. P., Buning, P. G.,and Elson, P. A., "PLOT3D User's Manual," NASA TM 101067, July, 1992.

[5]Soni, B. K., "Two- and Three-Dimensional Grid Generation for Internal Flow Applications of Computational Fluid Dynamics," AIAA 85-1526, 1985.

[6]Steinbrenner, J. P., Chawner, J. R., and Fouts, C.L., The GRIDGEN 3D Multiple Block Grid Generation System," Wright Research and Development Center Report WRDC TR90-33022, October, 1989.

[7]Thomas, P. D., Middlecoff, J, F., "Direct Control of the Grid Point Distribution in Meshes Generated by Elliptic Equations," AIAA Journal, vol 18, pp, 652-656, June, 1979.

[8]Erlich, L. W., "An Ad Hoc SOR Method, "Journal of Computationall Physics, vol. 44, pp. 31-45, March, 1981.


**Table I. Face numbers.**

| Face no. n : | j | k | l | $\xi$ | $\eta$ | $\zeta$ |
|---|---|---|---|---|---|---|
| 1 | 1 | varying | varying | 0. | varying | varying |
| 2 | jmax | varying | varying | $\xi_{max}$ | varying | varying |
| 3 | varying | 1 | varying | varying | 0. | varying |
| 4 | varying | $k_{max}$ | varying | varying | $\eta_{max}$ | varying |
| 5 | varying | varying | 1 | varying | varying | 0. |
| 6 | varying | varying | $l_{max}$ | varying | varying | $\zeta_{max}$ |

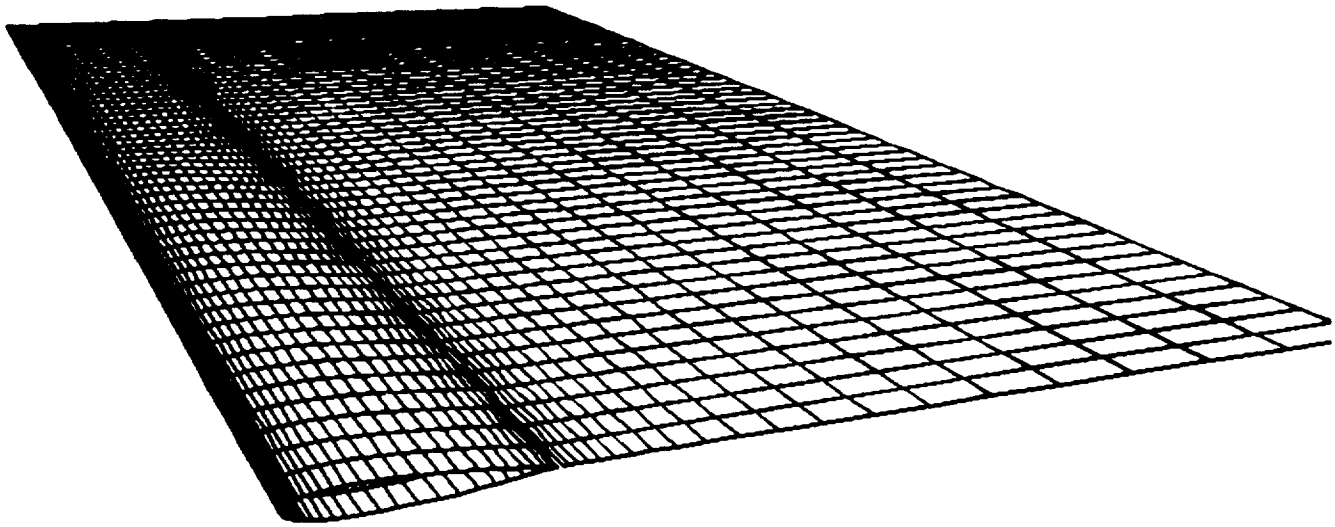


Figure 1a. Wing With Flat-Plate Extension in Both Spanwise and Downstream Directions.

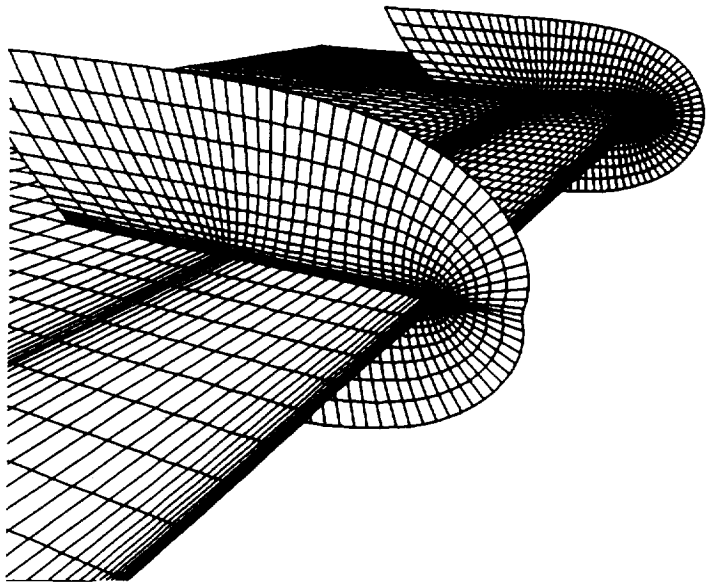


Figure 1b. View From Outboard End Showing Portions of Two Interior Grid Surfaces Wrapping Around Wing and Extension

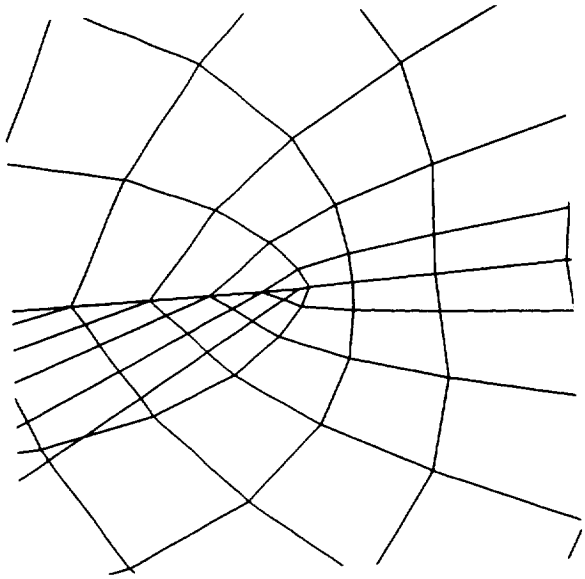


Figure 1c. Close-Up of Grid Surface Wrapping Around Leading Edge of Flat-Plate Extension

N95- 28751

# AUTOMATIC STRUCTURED GRID GENERATION USING GRIDGEN (SOME RESTRICTIONS APPLY)

John R. Chawner and John P. Steinbrenner
Pointwise, Inc.
Bedford, Texas

## OVERVIEW

The authors have noticed in the recent grid generation literature an emphasis on the automation of structured grid generation. The motivation behind such work is clear; grid generation is easily the most despised task in the grid - analyze - visualize triad of computational analysis (CA). However, because 1) grid generation is closely coupled to both the design and analysis software and because 2) quantitative measures of grid quality are lacking, "push button" grid generation usually results in a compromise between speed, control, and quality. Overt emphasis on automation obscures the substantive issues of providing users with flexible tools for generating and modifying high quality grids in a design environment. In support of this paper's tongue - in - cheek title, many features of the Gridgen software are described. Gridgen is by no stretch of the imagination an automatic grid generator. Despite this fact, the code does utilize many automation techniques that permit interesting regenerative features.

## INTRODUCTION

Gridgen [1] is a software system for generation of 3D, multiple block, structured grids. The system is comprised of two codes: Gridgen (see Figure 1), an interactive program that provides capabilities ranging from geometry model import through volume grid initialization and analysis software pre-processing; and Gridgen3D, a batch program for volume grid refinement. Gridgen is a monolithic program in the sense that it consists of a single process and a single graphics window. Gridgen's window, however, is repositionable and resizable. Gridgen also manages its own non-overlapping windows inside the main window. User interaction with Gridgen is directed through text - based button menus activated via the mouse or keyboard "hot keys". A 3D, graphical image of the grid is always present and may be easily panned, zoomed, rotated, and otherwise customized by the user.

The manner in which grids are constructed using Gridgen is is based on a hierarchical paradigm, beginning with the geometry model, and proceeding to curve, surface, and volume elements as shown in Figure 2. The geometry model, provided as input to Gridgen, serves as the foundation for the hierarchy. The user constructs curves which are in turn used to build the topological surface and volume components. The grid for each of the hierarchical components is an implicit but separate part of the component. The result of maintaining a distinction between the geometry model, the hierarchical components, and the grid allows changes to be rapidly propagated either forward or backward throughout the grid system.

Users operate Gridgen in much the same manner as a product designer operates a computer aided design (CAD) system. After importing the geometry model (database), the user draws curves (connectors), assigns a number of points to each connector, and distributes those points along the connector using a variety of distribution functions. The connectors are then selected as the boundaries of surface grids (domains). Transfinite interpolation (TFI), elliptic partial differential equation (PDE), and hyperbolic PDE methods are available for controlling the distribution of grid points on the domains. The domains are

then selected as the boundaries of volume grids (blocks). Again, TFI, elliptic PDE, and hyperbolic PDE methods are available for grid point control within the blocks. Finally, the user sets boundary conditions and exports formatted data for the chosen analysis software.

From the description above it should be apparent that grid generation using Gridgen is a user - in - the - loop task. Therefore, one of the goals of the software design has been to *automate* as much of the grid generation minutia and bookkeeping as possible so that the user may concentrate on topology and grid quality. This is made possible through Gridgen's data hierarchy. The data hierarchy maintains the inter-relationships between the 1D, 2D, and 3D grid components (connectors, domains, and blocks, respectively) so that the user's changes may be propagated up or down the hierarchy *automatically*. Some of these *automation* tools are described in the following sections.

## EXAMPLE GRID

Gridgen's *automation* features will be demonstrated in the context of grid generation for an external automotive shape. This bluff body, described in Reference [2] and illustrated in Figures 3, 4, and 5, was the subject of a wind tunnel test that studied its wake structure relative to the base slant angle. This vehicle was also used as the basis for an evaluation of computational fluid dynamics (CFD) software for automative shapes (Reference [3]). The blocking system and surface grids created using Gridgen are shown in Figure 6.

## DATABASE

Database is Gridgen's term for the geometry model of the object on and around which the grid is to be generated. Gridgen's database capability is based mathematically on $n$th degree, rational, Bezier curve and surface geometry. The database also includes relational data such as grouping, color, names, etc. Since Gridgen is dedicated to the generation of hexahedral grids, the user's CAD software may be used to create the database. Gridgen can import the database from several industry standard file types, including those listed below.

- PLOT3D (bilinear surfaces)
- Patran V2.5 Neutral[1]
- IGES[2]

The bilinear database surfaces for the bluff-body grid were created from the drawings and tabular data in Reference [2]. The surfaces in this particular database were stored in two files: one containing the five surfaces for the fore- and mid-bodies, and one containing the four aft-body surfaces.

Gridgen offers several methods for placing grid points on the database including curves that may be drawn directly on the surfaces ("Line on DB" and "Curve on DB" segment types), projection of curve grids, and projection of surface grids. More importantly, Gridgen *automatically* maintains the adherence of grid components to the database. This relationship between the grid and the database is maintained via the database entity name. Each entity in Gridgen's database has a unique name and every grid component

[1] Packet types 32 and 33
[2] Entities 000, 100, 102, 104, 106, 110, 116, 124, 126, 128, 212, 308, 314, 402, 406, 408

that adheres to the database saves the name of the entity and the database coordinates (see Figure 7). The utility of this feature is explained in the following paragraphs.

Gridgen provides several commands that allow changes to be made interactively to the database. The entire list of database commands is listed below.

- Import
- Export
- Examine
- Name
- Group
- Ungroup
- Reduce
- Copy
- Delete
- Translate
- Scale
- Rotate
- Intersect

When the shape modification commands (Translate, Scale, and Rotate) are applied by the user, all of the grid elements adhering to the modified entities (as determined by the data stored in Figure 7) are also modified *automatically.*

Changes to the grid based on parametric variations of the database can also be made in a "hands off" mode. As before, let's assume that a grid for the 40° base slant has already been created and that we wish to create a grid for the 20° base slant. Recall that Gridgen associates grid components with database entities via the entity name. Our goal, therefore, will be to create a database for the 20° base slant case and have Gridgen think that it's the same as the 40° configuration. In order to accomplish this it's important to understand Gridgen's convention for naming database entities. For IGES files it's simple: Gridgen uses the names stored in the file. For PLOT3D and Patran files, which don't contain entity names, Gridgen constructs the name from a text string of the form `basename-type-number` where

`basename` is the basename of the file from which the entity was imported,
`type` is a descriptive term for the type of entity (e.g, `psurface` for a parametric surface),
`number` is the sequence number of the entity in the file

Since the user knows that a variation of the base slant angle will require new several aft databases, the database files can be created ahead of time and named `aftxx.net` where the `xx` is the base slant angle, ie `aft40.net`. When the baseline 40° case is first generated we copy the 40° database file to a generic file

```
cp aft40.net Aft.net
```

Then, the baseline grid will be generated with references to database surfaces called `Aft-psurface-n` and the grid is saved to a Gridgen file. In order to generate grids for the other base slant angles the following steps can be used.

1. Copy the 20° database file to the same generic name used for the baseline case, ie.

   ```
   cp aft20.net Aft.net
   ```

2. Import into Gridgen the database files `Body.net` and `Aft.net`.
3. Import the Gridgen file that was created for the 40° case.

Since Gridgen doesn't notice a difference between the current and previous database configuration (ie, a database entity called `Aft-psurface-1` has been imported and there are grid references to that entity) the

grid is *automatically* regenerated in terms of the new database through the simple act of importing the files. Therefore, with a little forethought, the user can parametrically change the shape of the database without requiring significant grid rework. An example of this is shown in Figure 8 which shows how the 40° grid was changed into the 20° grid simply by importing a new database file.

## CONNECTORS

The one dimensional grid element in Gridgen's data hierarchy is the connector. Each connector consists of three sets of information: the 3D curve shape, the number of points on the connector (dimension), and parameters describing the distribution of grid points along the connector.

The shape is defined interactively by how the user draws the connector. Each connector is made up of one or more segments where a segment is a primitive curve type including those listed below.

- Line (a 3D polyline)
- Curve (a 3D cubic Bezier curve)
- Akima Curve (a 3D cubic Bezier curve with slopes defined via Akima's method)
- Line on DB (same as a Line but constrained to lie on a database surface)
- Curve on DB (same as a Curve but constrained to the database)
- Akima Curve on DB (same as an Akima Curve but constrained to the database)
- Circular Arc
- Conic Section

Mathematically, the segments are represented by cubic, rational, Bezier curves. This underlying similarity allows the segments to be converted with a single Gridgen command from a Line, to a Curve, to an Akima Curve, and even to a generalized Bezier segment as shown in Figure 9. The Bezier segment affords the user the opportunity to edit the slope point of the Bezier control polygon. This flexibility is provided so that the Gridgen user can create connectors with the desired shapes. Editing control points is only one of Gridgen's connector modification commands which are listed below.

- Add Segment
- Insert Segment
- Erase Segment
- Edit Control Points
- Translate
- Stretch
- Rotate
- Project onto DB
- Split
- Join

The user assigns a number of grid points (called the dimension) to each connector by either typing a numerical value or by graphically picking a series of existing connectors and using the sum of their dimensions for the current connector.

The distribution of grid points along the connector is defined by several parameters, some of which are shown in Figure 10. Breakpoints are locations along the connector at which a grid point will lie and at which the grid point spacing will be explicitly defined by the user. Breakpoints also divide the connector into subconnectors. In each subconnector the user controls the number of grid points (subject to the total connector dimension), clustering at each breakpoint, and distribution function type which may be selected from this list:

- hyperbolic tangent
- monotonic rational quadratic spline
- geometric progression
- copied from other connector(s)

Although each of the three attributes of a connector (shape, dimension, distribution) are specified separately, they are all coupled. This allows the user to change any one of the attributes and the other two will be updated *automatically.* This coupling of the connector's attributes allows the user to edit the grid without reworking the whole connector which can make the user much more efficient since creating connectors is the most labor intensive task in Gridgen. This connector updating process occurs whenever the database is modified (as described in the previous section). As will be shown in the following sections, changes to the connectors are propagated *automatically* up the grid component hierarchy to domains and blocks.

## DOMAINS

Surface grids (domains) are created by the user's selection of the connectors that comprise the domain's perimeter. [3] Any number of connectors may be used on the perimeter but they must be arranged into four edges such that opposite edges have the same number of grid points (ie, the domain must map into an $I \times J$ computational rectangle). Once the domain perimeter has been defined Gridgen will *automatically* create the surface grid points by applying an algebraic grid method called transfinite interpolation (TFI). This method uses the known grid point data $\vec{r} = [\ x\ y\ z\ ]^T$ on each connector (obtained from each connector's grid point distribution) to interpolate $\vec{r}$ at each interior grid point. Furthermore, when Gridgen detects that all of the connectors used to bound the domain lie on the same database surface the TFI algorithm will be applied *automatically* in the parametric space of the database surface. In this case, parametric TFI uses boundary data of the form $\vec{u} = [\ u\ v\ ]^T$ where $u$ and $v$ are the parametric (domain space) coordinates of the database surface (see Figure 7). This interpolation yields $(u, v)$ coordinates at each interior grid point which are then evaluated in terms of the database surface resulting in $\vec{r} = [\ x\ y\ z\ ]^T$. Parametric TFI ensures that the surface grid points will adhere to the database without any need for projections.

This same type of *auto*-TFI logic is applied when connectors that belong to domains or the domains themselves are modified. Domain modification commands include those shown below.

- Translate
- Stretch
- Rotate
- Project onto DB
- Split
- Join

The new connector shape change is used to re-TFI the domain surface grid *automatically* using whatever TFI method was last applied to the domain by the user. This type of modification, where changes propagate up Gridgen's data hierarchy (see Figure 2) is called forward editing.

Surface grid quality (smoothness, clustering, orthogonality) can be improved by application of Gridgen's elliptic PDE techniques. Specifically, Gridgen solves Poisson's equation using an explicit, point-wise, successive over relaxation (SOR) algorithm with optimal relaxation factors ([4]) subject to user selected control functions. The elliptic PDE method is quite flexible and allows the user to apply a wide variety of attributes, some of which are listed below.

[3] Gridgen can also create a domain by applying a hyperbolic PDE technique to a connector.

- Control Functions
  - LaPlace (smoothness)
  - Fixed Grid (smoothness)
  - Thomas-Middlecoff (clustering) [5]
  - Sorenson (orthogonality) [6]
  - Hilgenstock-White (orthogonality) [7] [8]
- Boundary Conditions
  - Fixed
  - Float
  - Ortho
- Surface Shape
  - free
  - fixed
  - database
- Numerical Solution
  - relaxation factor

One of the most important elliptic PDE attributes is the surface shape. This attribute in combination with the control functions allows the grid points to redistribute themselves in order to obtain the desired grid qualities while also adhering to the desired shape. Of course, the most important shape that users need to maintain is the shape of surface grids constrained to the database. There are two techniques with which Gridgen can make grid points adhere to the database. As was the case for TFI, if Gridgen determines that the surface grid points lie on the database, the elliptic PDEs will be solved in the parametric space $(u, v)$ of the database and the model space coordinates will be obtained from the known surface shape $\vec{r} = f(u, v)$ (parametric technique). The second technique for keeping grid points on the database is a simple projection (conventional technique). Each domain being run in the elliptic PDE solver maintains the database entities to project onto, and the projection orientation (which is computed automatically by Gridgen). The advantage of the parametric elliptic PDE technique is that it's much easy to set-up (no additional attributes) and it's much faster (no projections). Fortunately, Gridgen has *automated* the application of the database surface technique. Specifically, when Gridgen detects that all of a grid point's neighbors (those points used in the finite differences for solution of the elliptic PDEs) lie on the same database surface the code will *automatically* apply the parametric technique, thus making the calculation much more efficient. When domains span multiple database surfaces, only those grid points that lie on or near the seam between the surfaces will require projection. Of course, grid points may migrate across the seam from one surface to another. Figure 11 shows a surface grid that has been created on the surface of the bluff body database spanning two forebody surfaces with notations as to which grid points are solved parametrically and which are solved conventionally.

It is often the case that the precise shape of a surface grid's perimeter (the connector) is not known by the user. This is especially true for connectors shared between two domains. Even when shape isn't important the requirement for smooth variation of grid lines across the connector is required. Rather than forcing the user to choose between 1) manually fine-tuning the connector's shape or 2) being stuck with something inadequate, Gridgen's elliptic PDE techniques provide a boundary condition that allows the connector shape to float subject to the elliptic PDE solution. The float boundary condition treats the grid points on a connector shared between two domains in the same manner as interior domain points; they move subject to the elliptic PDE solution. When the solver is done the original connector shape is replaced by a new shape defined by the refined grid points. An example of this feature is shown in Figure 12.

The user may also defined sub-domains which, as the name implies, are subsets of a domain. Subdomains may then be used with the TFI and elliptic PDE methods to restrict the effects of the methods to a small region of the grid. Subdomains may be thought of as templates that fit over the domain grid and define the region to which changes should be made. Each subdomain saves its own unique combination of elliptic PDE attributes. When grid modifications require re-application of the PDE methods (see previous sections) the user may simply invoke the PDE method without resetting attributes.

## BLOCKS

Volume grids (blocks) are created by the user's selection of the domains that comprise the block's perimeter. [4] Any number of domains may be used on the block's perimeter but they must be arranged into six faces such that opposite faces have the same number of grid points (ie, the block must map into an $I \times J \times K$ computational parallelepiped). Once the block perimeter has been defined Gridgen will *automatically* create the volume grid points by applying TFI.

Block modifications, including copy, translate, scale, and rotate, will result in *automatic* propagation of changes down Gridgen's data hierarchy (backward editing) to the domains and connectors.

Gridgen has also *automated* one of the most cumbersome aspects of multiple block grid generation: interblock connectivity detection. As a result of Gridgen's data hierarchy, the code *automatically* detects and maintains both full- and partial-face interblock connections simply by noting that the same domain is used in two blocks. Changes to a domain's grid points by the PDE solver, for example, are *automatically* transferred to both blocks that use that domain. Changes to a domain on a block's perimeter will propagate up Gridgen's data hierarchy and cause *automatic* re-TFI of the volume grid points. Also, when the boundary conditions are exported for use with the analysis software the connectivity data is formatted and exported specifically for the chosen analysis software.

Gridgen also offers an interactive tool for re-dimensioning (changing the total number of grid points) the entire blocking system. This feature allows the user to select any connector in the blocking system and change its number of grid points. That change is propagated *automatically* throughout the system to maintain dimensional consistency (ie, maintain $I \times J$ domains and $I \times J \times K$ blocks).

## MISCELLANEOUS

One final *automation* feature deserves mention. Gridgen can import "raw" grid points (from a PLOT3D grid file, for example). When this is done, Gridgen *automatically* creates connectors, domains, and blocks, from the raw grid data. Therefore, the user may then avail him/herself of all of Gridgen's commands, including analysis boundary condition setting and file export. Conversely, the Gridgen user may export the grid points from any connector, domain(s), or block(s) to a PLOT3D file for use in other applications.

## SUMMARY

Various features within Gridgen have been described and their unique *automatic* aspects have been emphasized. These features contrast sharply with the fact that Gridgen is not an *automatic* grid generator.[5] It is not the intent of this paper to diminish the many published and unpublished works on automatic grid generation. Indeed, the successful application of CFD in a design environment often depends on configuration - limited procedures that may be applied rapidly and easily by the CFD novice. Instead, this paper and Gridgen itself have been motivated by the desire to emphasize grid quality and software

[4] Gridgen can also create a block by applying a hyperbolic PDE technique to a domain.

[5] We define an automatic grid generator as one which will create a suitable grid given a geometry model and little or no user input.

flexibility. These emphases have led to the development of a visually based, 3D, interactive application environment that *automates* much of the low level grid maintenance tasks, freeing the user to apply his/her judgement as to grid suitability. Gridgen's ability to *automate* much of the process is directly attributable to its data hierarchy in which the curve, surface, and volume grids are maintained as secondary components of connector, domain, and block topology components.

## ACKNOWLEDGEMENTS

Gridgen is a product of Pointwise, Inc. GRIDGEN Version 9 was sponsored by NASA Ames Research Center through Computer Sciences Corp. Thanks to Mr. Matthew Blake of NASA and Dr. Jin Chou of CSC. GRIDGEN Version 8 was sponsored by NASA Langley Research Center through Computer Sciences Corp. Thanks to Dr. Robert Smith of NASA and Dr. Jamshid Abolhassani of CSC. GRIDGEN Version 6 was sponsored by the United States Air Force Wright Research and Development Center. Thanks to Dr. Donald Kinsey and Lt. John Seo. All trademarks used herein are the property of their respective owner.

## REFERENCES

[1] Steinbrenner, J.P., and Chawner, J.R., "The Gridgen User Manual: Version 10", available from Pointwise, Inc., Jan. 1995.

[2] Ahmed, S.R., Ramm, G., and Faltin, G., "Some Salient Features of the Time - Averaged Ground Vehicle Wake," SAE Technical Paper Series No. 840300, SAE International Congress and Exposition, Detroit, MI, March 1984.

[3] Misegades, K.P., "1988 Automobile Aerodynamics Workshop," Second International Conference on Supercomputing in the Automotive Industry, Seville, Spain, October 1988.

[4] Ehrlich, L.W., "An Ad Hoc SOR Method", **Journal of Computational Physics**, Vol. 44, pp. 31-45, 1981.

[5] Thomas, P.D., and Middlecoff, J.F., "Direct Control of the Grid Point Distribution in Meshes Generated by Elliptic Equations", **AIAA Journal**, Vol. 18, pp. 652-656, 1979.

[6] Sorenson, R.L, "The 3DGRAPE Book: Theory, Users' Manual, Examples", NASA TM-102224, July 1989.

[7] Hilgenstock, A., "A Fast Method For The Elliptic Generation of Three-Dimensional Grids With Full Boundary Control", **Numerical Grid Generation in Computational Fluid Mechanics '88**, ed. by Sengupta, S., et al, Pineridge Press Ltd., Swansea, UK, 1988, pp. 137-146.

[8] White, J.A., "Elliptic Grid Generation With Orthogonality and Spacing Control On An Arbitrary Number of Boundaries", AIAA paper no. 90-1568, June 1990.

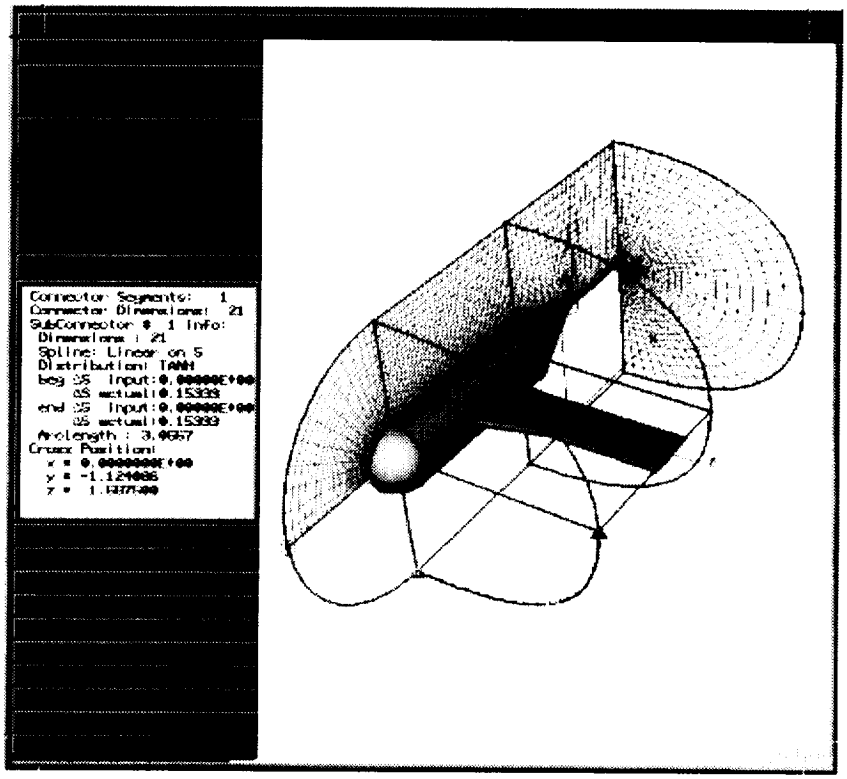


Figure 1: The Gridgen screen.

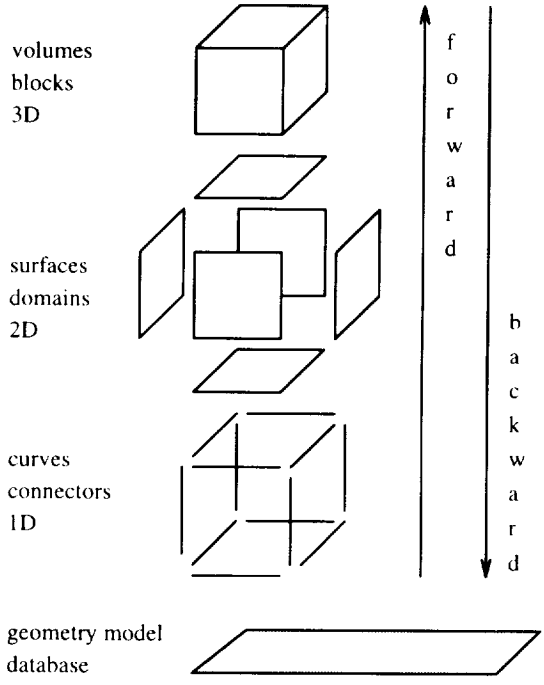



Figure 2: Gridgen's data hierarchy.

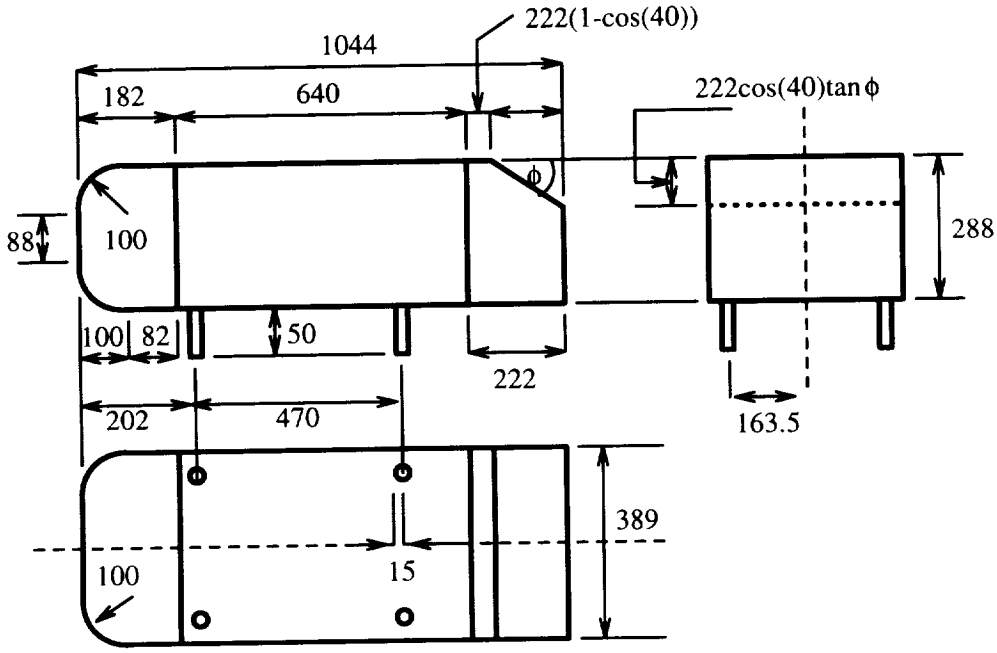
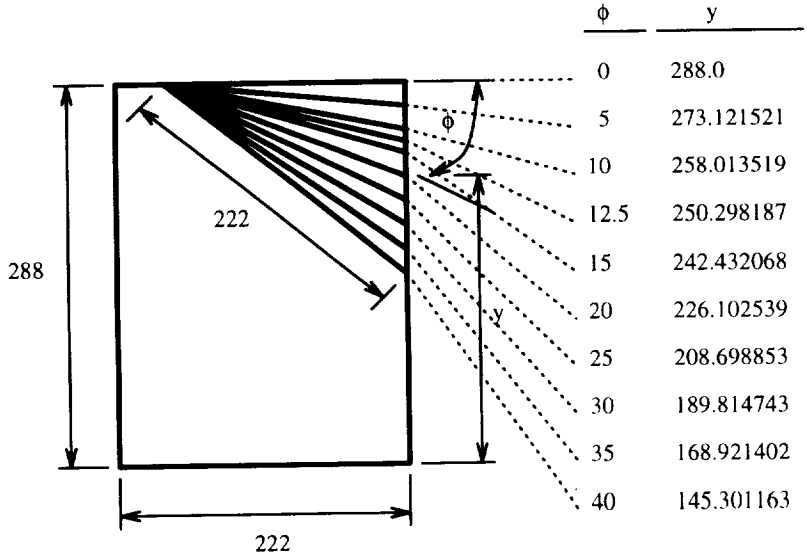



Figure 3: Schematic of a bluff-body, external automotive shape used for demonstration of Gridgen's automation features. All dimensions in millimeters. Adapted from Reference [2].




| ϕ | y |
|---|---|
| 0 | 288.0 |
| 5 | 273.121521 |
| 10 | 258.013519 |
| 12.5 | 250.298187 |
| 15 | 242.432068 |
| 20 | 226.102539 |
| 25 | 208.698853 |
| 30 | 189.814743 |
| 35 | 168.921402 |
| 40 | 145.301163 |

Figure 4: Detailed schematic of the various base angles for the bluff body in Figure 3. All dimensions in millimeters. Adapted from Reference [2].

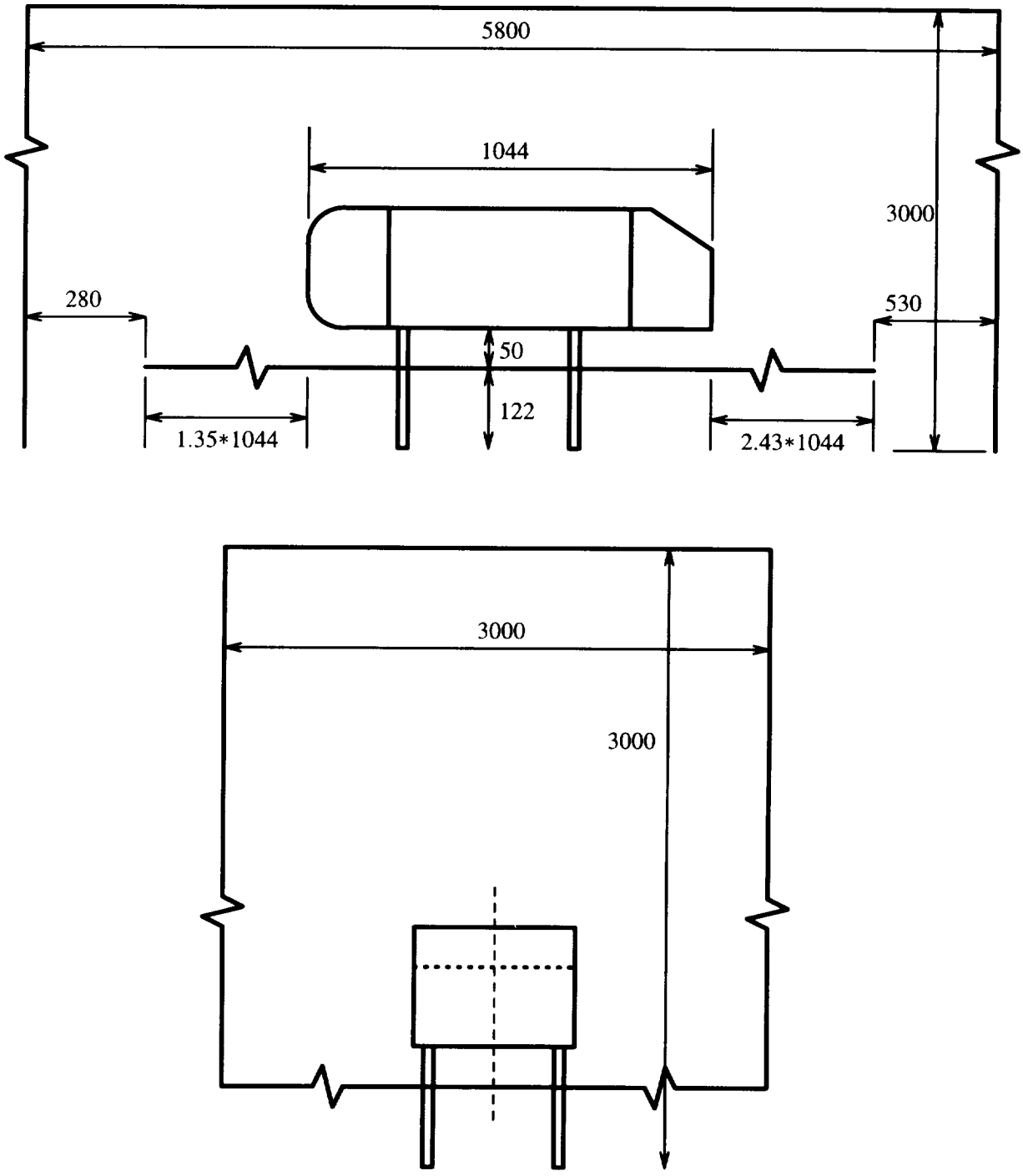



Figure 5: Schematic of the bluff body in Figure 3 installed in the wind tunnel. All dimensions in millimeters. Adapted from Reference [2].

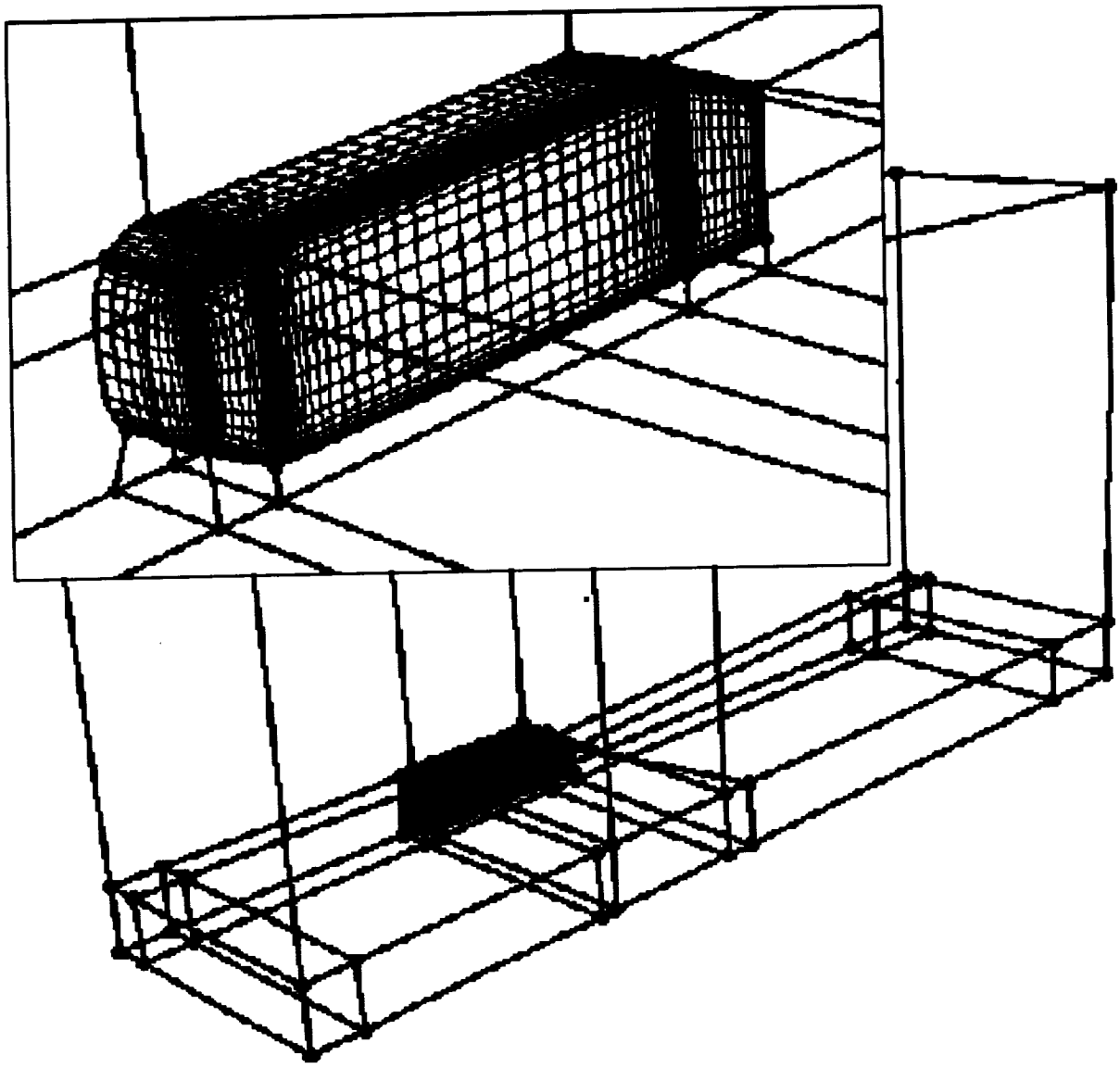


Figure 6: A 15 block grid containing 520,000 grid points was created for the bluff body. A close-up of grids on the surface of the body is also shown.

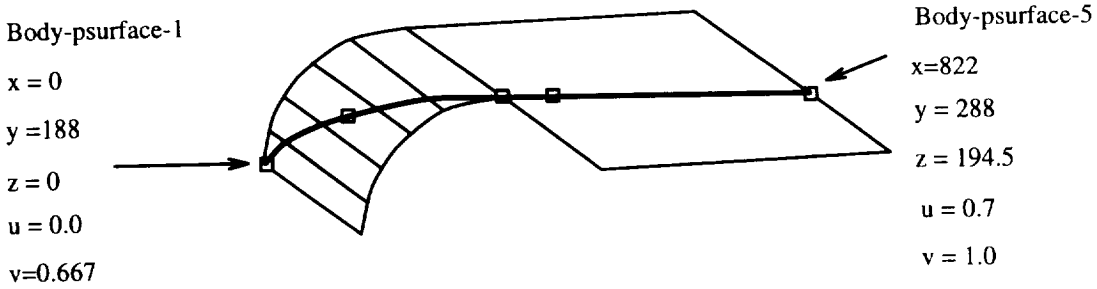



Figure 7: Gridgen maps grid elements to the database via the entity name allowing the underlying shape to be easily changed.

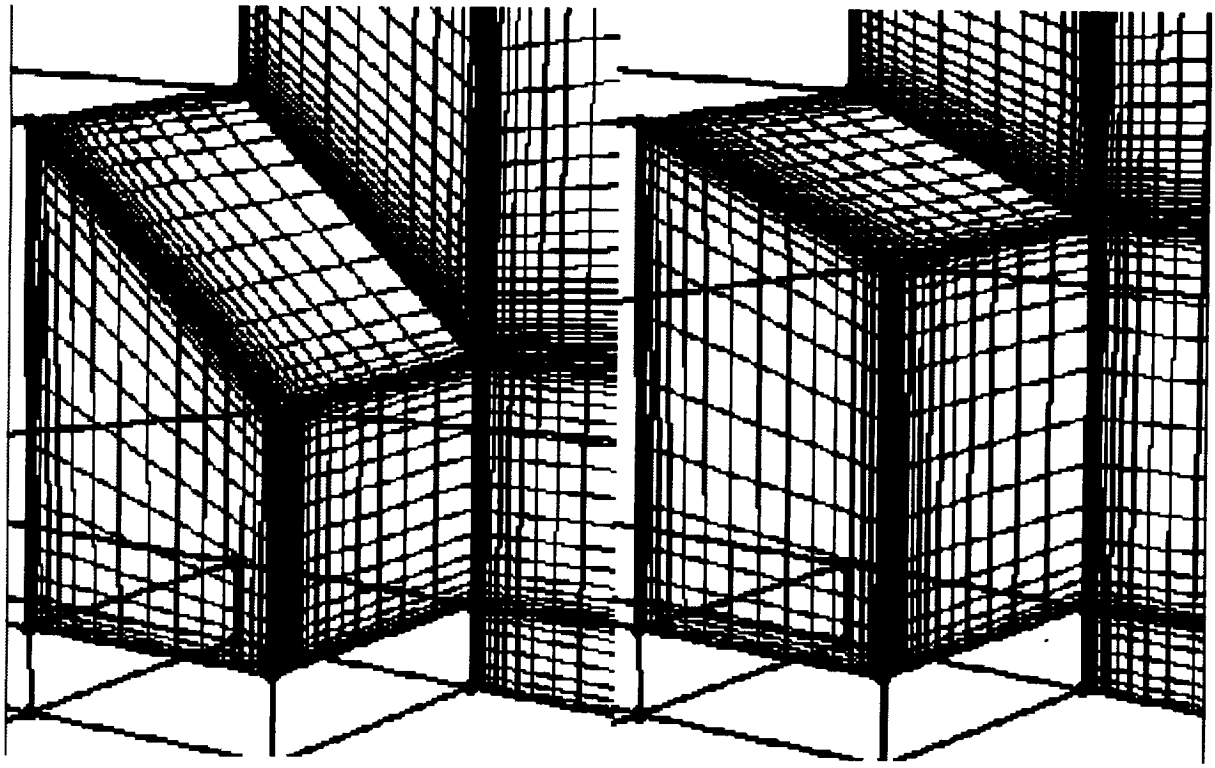


Figure 8: The grid for the 40° base slant angle (left) was changed to the 20° base slant (right) simply by importing a different database file.

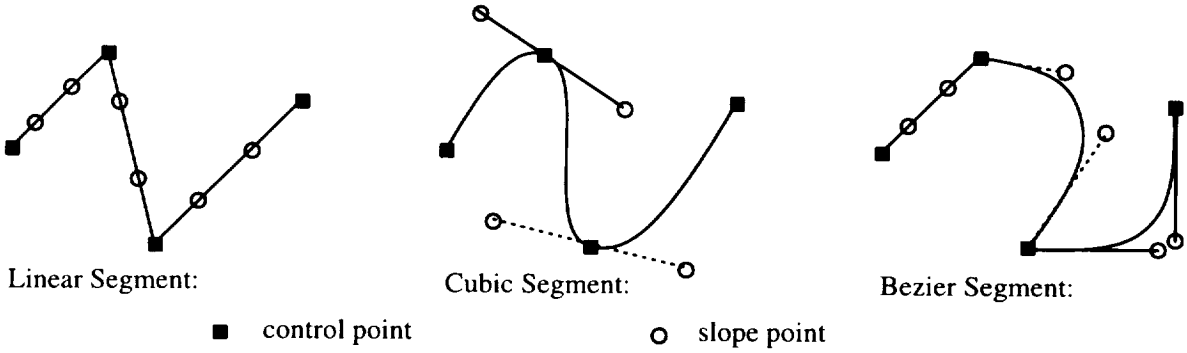



Figure 9: Gridgen's unified geometric foundation allows segments to be easily converted from one type to another. Note that the control points for all three segments are the same.

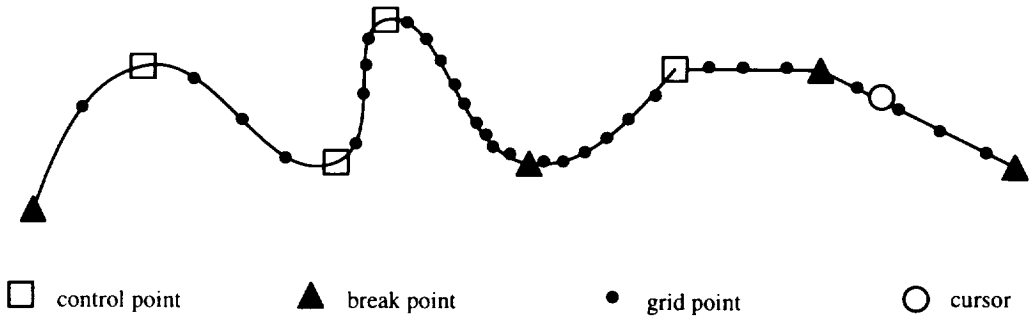



Figure 10: Terminology used in the distribution of grid points along a connector.

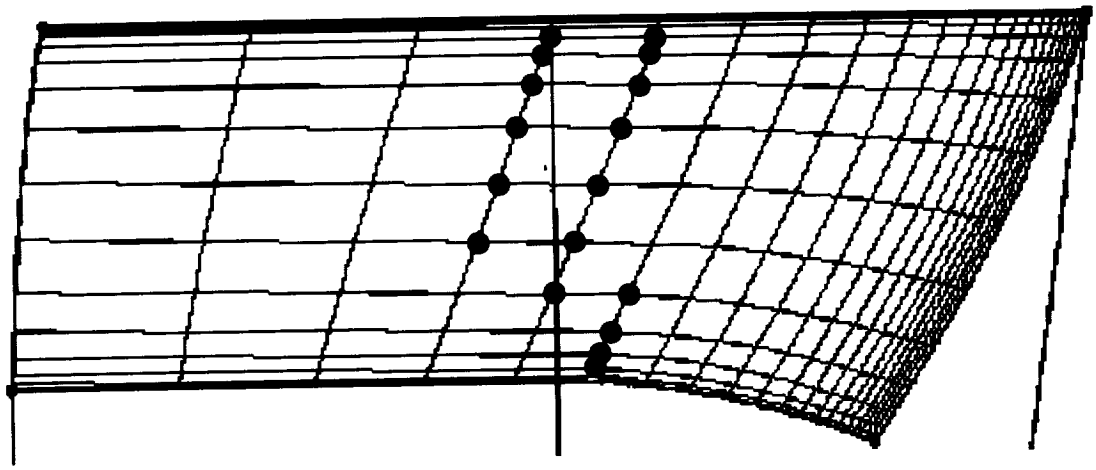


Figure 11: The surface grid on the upper forebody spans two database surfaces (the vertical lines extending beyond the grid are the boundaries of the surfaces). The filled circles denote grid points that are projected onto the surfaces. All other points are maintained on the surfaces parametrically.

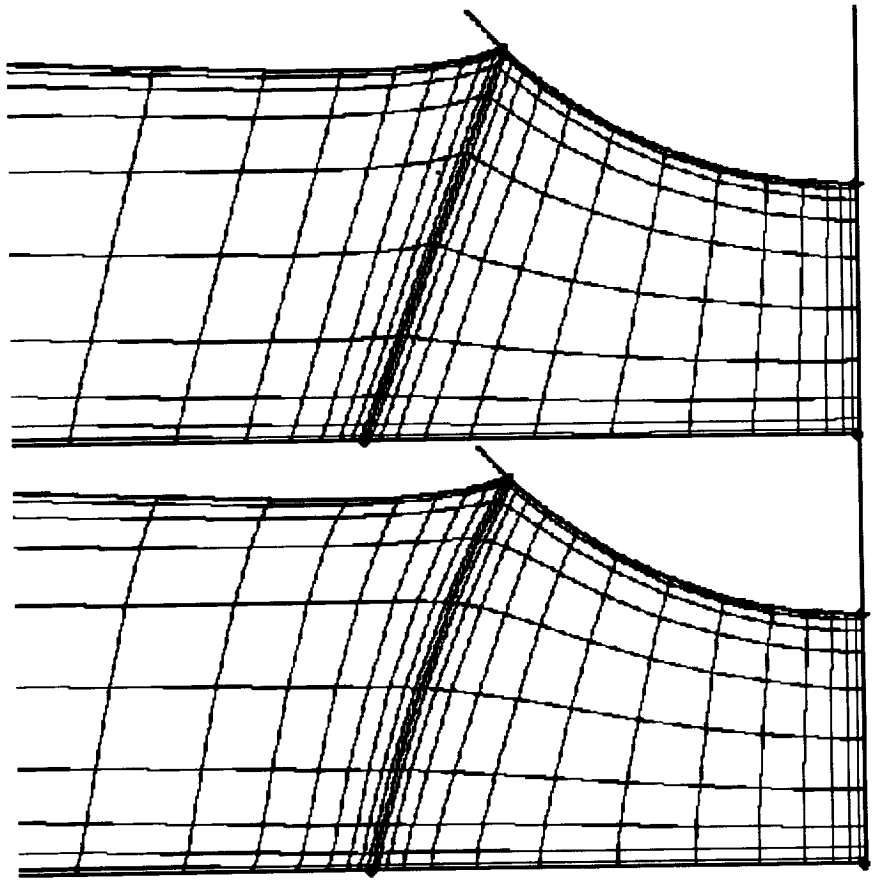


Figure 12: The connector between two surface grids may change shape according to the elliptic PDE solution.

# TUNED GRID GENERATION WITH ICEM CFD

Armin Wulf and Vedat Akdag
ICEM CFD Engineering
Berkeley, CA 94704

## ABSTRACT

*ICEM CFD is a CAD based grid generation package that supports multiblock structured, unstructured tetrahedral and unstructured hexahedral grids. Major development efforts have been spent to extend ICEM CFD's multiblock structured and hexahedral unstructured grid generation capabilities. The modules added are: a parametric grid generation module and a semi-automatic hexahedral grid generation module. A fully automatic version of the hexahedral grid generation module for around a set of predefined objects in rectilinear enclosures has been developed. These modules will be presented and the procedures used will be described, and examples will be discussed.*

## INTRODUCTION

The ability to accurately create a computational grid about geometrically complex configurations is becoming increasingly important in the analysis world. With ICEM CFD computational grids can be employed to treat complex geometric topologies. ICEM CFD embodies full CAD tools for creating geometries or importing geometry from various CAD systems. Computational grids, including boundary conditions can be generated for over 25 different CFD flow solvers and structural analysis codes [1]. (Figure 1).

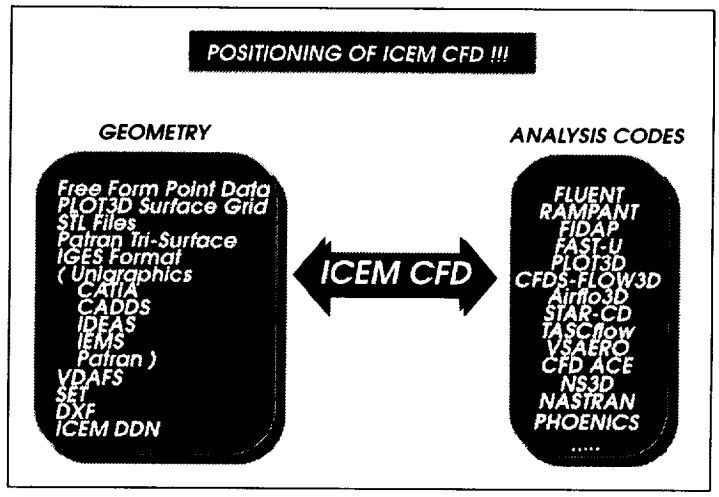



*Figure 1: Positioning of ICEM CFD between the geometry and analysis codes.*

Existing methods for multiblock structured computational grid generation codes are generally very time consuming. Current codes also require high level of user expertise in order to achieve optimal usage. Since the rapid construction of suitable multiblock structured computational grids is still one of the pacing issues in CFD applications, additional functionality has been added to ICEM CFD. They are ICEM COMAK for parametric multi-block mesh generation and ICEM HEXA for semi-automatic hexahedral mesh generation. In addition, the fully automatic hexahedral grid generation module called ICEPAK will be presented. ICEPAK has been developed in coorporation with Fluid Dynamics International (fdi) [2] to support thermal management of electronic enclosures. For ICEPAK fully automatic grid generation is possible, because the geometry representation is restricted.

## PARAMETRIC GRID GENERATION USING ICEM COMAK

ICEM COMAK is a parametric multi-block grid generation tool which is an extended option of ICEM CFD's structured grid generation code. Once the mesh of a given configuration is created, using COMAK (COnfiguration MAKer) it is possible to replay the construction process in order to get a mesh of the same topology with geometric differences. It offers two modes; the Specifica-

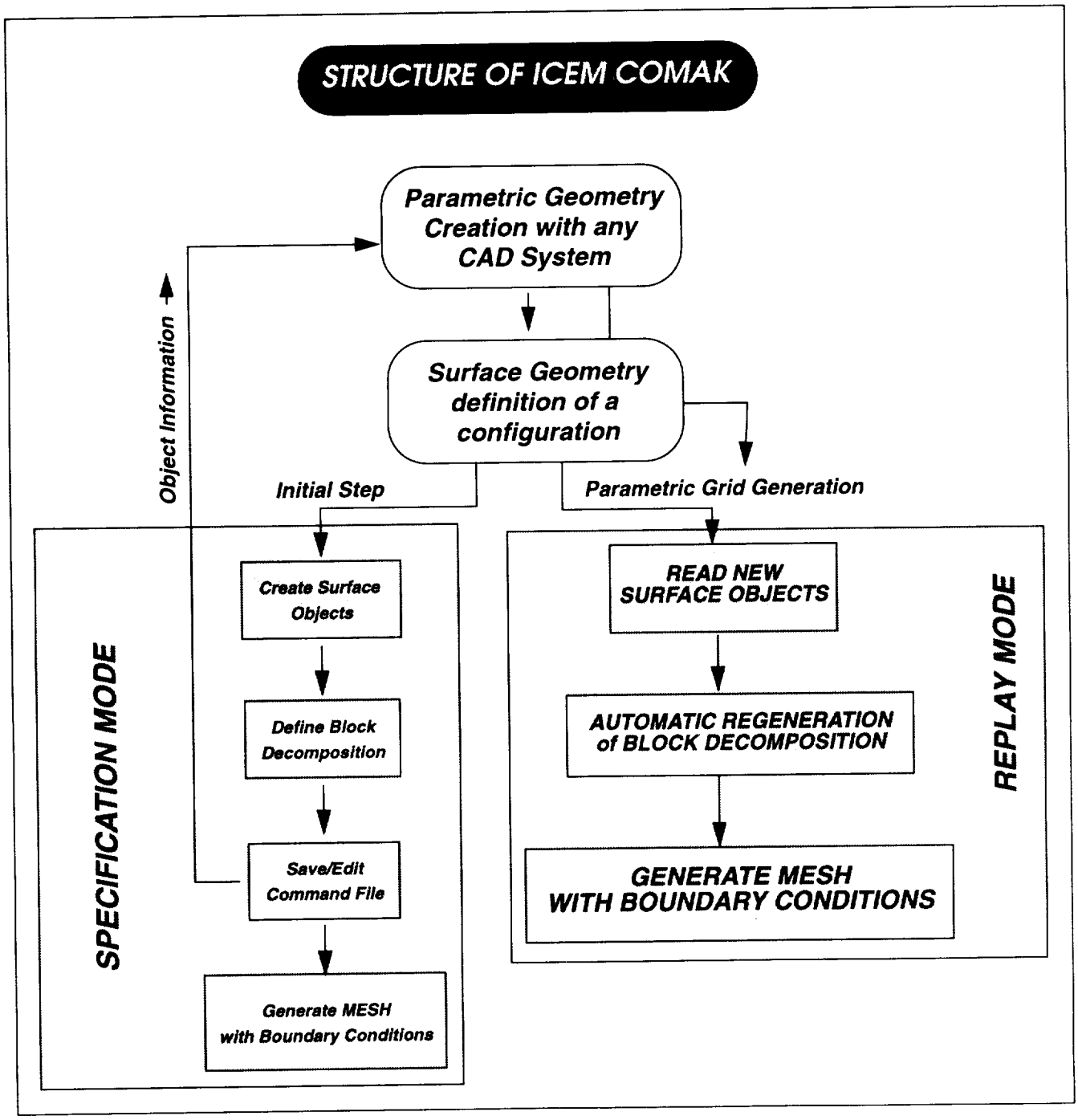



*Figure 2: Structure of ICEM COMAK, the parametric grid generator*

tion Mode and the Replay Mode. In the Specification Mode the user has a set of tools allowing him to interactively create, manipulate, group and manage geometric entities. The user's actions are stored into a command file. In the Replay Mode, when the user wants to create a mesh around a different object with the same topology, it is only necessary to break the object geometry into parts like the original. Then the user can replay the commands to create the new multi-block structured mesh associated to the new CAD geometry, where sets of geometrical entities can be different. The boundary conditions are set-up automatically. These features allow one to perform very efficient geometric trade studies.

In Figure 2, the structure of ICEM COMAK is shown. CAD systems such as I/EMS from Intergraph or Pro-Engineer from Parametric Technology Corporation are modelers that can create geometries using a parametric approach.

During the initial step or the specification mode, the geometry from the user's CAD system is input in IGES format to the ICEM COMAK environment. The first step is to put surfaces into object groups. The user then composes the computational block structure. During this operation the session commands are recorded in a command file. The file created is saved. After the points along block edges are distributed, the computational grid is created. The boundary conditions for the analysis are also created during this session. User then translates the mesh into the flow solver format of choice.

To perform the parametric mesh calculation for a similar geometry with the same topology but with geometric differences, the user goes back and names the objects in the new geometry with the same object names. After the new IGES file is created, the ICEM Manager grabs this IGES file and writes the CAD data into the object database and initiates ICEM COMAK. COMAK reads the new object files and automatically regenerates the computational grid by updating the computational topology with the new geometry and translates the mesh into the flow solver format. The replay mode can be treated as a continuous loop for computational grid generation in batch mode on modified geometries.

The ICEM Manager is written in TCL/TK [3] a programming system for developing and using graphical user interface applications. It is an easy to use scripting language for controlling and extending applications. The following sample script illustrates the execution of a series of commands in the ICEM Manager to support the automatic replay mode as it is described above. In this example it creates an input for the structural analysis code ANSYS from Swanson Analysis Systems, Inc.

```
# This script shows how to run a complex command from within the manager.
# The actions it performs are:
# 1. Translates two IGES files into DDN parts. The names of these parts
# and the IGES files are known beforehand.
# 2. Run a GPL program on each part, which extracts a set of Comak
# objects.
# 3. Run a Comak script to automatically generate the mesh.
# 4. Write an ANSYS output.

# To make this script available, you should put the following line in your
# ~/.iceman_init file:
#
# if {file exists user_app_script.tcl} { source user_app_script.tcl }
```

```
#
# The rest of this file is the contents of custom_script.tcl, which
# should be in the directory you run icemcfd from.

proc user_app_script {what} {
 global env ddn_path confname geoname partname s2u_path ansys_out-
put_path
 global partname sheetname

 if {($what == "all") | | ($what == "prepare")} {
# First remove all the existing DDN parts and the Comak objects.
set p (glob -nocomplain parts/* objects/$confname/$geoname/*)
if {$p != ""} {
 eval exec /bin/rm -f $p parts/.ddn_directory \
 objects/$confname/$geoname/.ddn_directory
}
update_partlist d 1

# Set up the IGES directive file.
exec /bin/echo "CONVERT,NAME > _directive_tmp
exec /bin/echo "define,CREATE_310_PART=1 >> _directive_tmp

# Convert all the parts in the iges directory to DDN parts.
# If any of them have "name" in their filename, run them through
# the gpl program.
set names ""
foreach if (glob iges/*) {
 # Convert the part to IGES.
 set dn (file tail (file rootname $if))
 set command "$env(ICEM_ACN)/iges/iges_post i=iges/$pf name=$dn \
 o=parts/$dn l=iges_list logit=yes d=_directive_tmp"
 runcom $command iges_list
 if (string match "name" $dn) { lappend names $dn }
}
update_partlist d 1

# Make a DDN command file that runs the GPL program and then exits.
set restart (open .restart.tmp w)
puts $restart "F.5.13.5.3.(pwd)"
puts $restart "F.5.13.3.kim1_gpl"
puts $restart "F.4.7.y"
close $restart

foreach pf $names {
 # Run the GPL program.
 set_part_partfile parts/$pf
 if {$partname == ""} { error "No part" }
 set command "$ddn_path db=parts pn=\"$partname\" sn=1 \
 i=.restart.tmp (get_ddn_defaults)"
 runxcom USER_APP_GPL $command "" 0 0
 update_partlist d 1
}

# Now move all the parts to the Comak directory.
foreach pf (glob parts/*) {
 if (string match "name" $pf) continue
 exec /bin/cp $pf objects/$confname/$geoname
}
update_partlist k 1

# Clear out the old domains.
set doms (glob -nocomplain mulcad/$confname/$geoname/domains/*)
if {$doms != ""} { eval exec /bin/rm -f $doms }
 }

 if 0 {
if {($what == "all") | | ($what == "comak")} {
 set extras ""
 # Now run the Comak job. These key sequences are the commands that
the
 # user would type in to Mulcad to perform the indicated actions.
{ 52))Run the key.cmd file. }
{ nnnDon't modify parameters. }
{ ))yExit comak. }
{ 03yy)Update topology. }
{ !253yMesh generation. }
{ )))Back to the top. }
{ 96yExit mulcad. }
 } {
append extras (lindex $xx 0)
 }

 unselect_part
 catch {exec /bin/rm -f (glob mulcad/$confname/$geoname/parts/test1*)}
 update_partlist m 1
 set partname TEST1
 set sheetname 1

 set key (send_keyboard_events "ICEM 3.1 GRAPHICS" 5 $extras)
 app_mulcad
 exec /bin/rm -f $key
}
 }

 if {($what == "all") | | ($what == "comak")} {
unselect_part
catch {exec /bin/rm -f (glob mulcad/$confname/$geoname/parts/test1*)}
update_partlist m 1
set partname TEST1
set sheetname 1

#1.1.12.).)
set extras {16 5.2.).) ).) y 10.3 Y Y n ) 11.2.1.12.).5.3}
app_mulcad $extras
 }

 if {($what == "all") | | ($what == "output")} {
# Convert the structured domains into an unstructured one.
set doms (glob mulcad/$confname/$geoname/domains/*)
set topofile mulcad/$confname/topology/topo_mulcad_out
set outfile $place/domains/struct_merge
runcom "$s2u_path -t $topofile -o $outfile $doms"

# Now run the ansys converter.
set topofile mulcad/$confname/$geoname/boco
set ansys mulcad/$confname/$geoname/transfer/ANSYS
runcom "$ansys_output_path -dom $outfile -b $bocofile $ansys"

infomsg "Done with conversion."
 }
}

# These lines create a new menu and add the above command to the menu.

make_menu user_app "USER Application" ""
make_entry user_app "Whole script" "" "" "user_app_script all" ""
make_entry user_app "Prepare objects" "" "" "user_app_script prepare" ""
make_entry user_app "Run Comak" "" "" "user_app_script comak" ""
make_entry user_app "Write output" "" "" "user_app_script output" ""
```

Figure 3 illustrates the application of ICEM COMAK to full airplane volume grid configurations [4]. Using the specification mode the volume grid is calculated for the A320 aircraft. The command file is replayed to create the similar grid for the A330 aircraft. Notice the differences in the sizes of the engine and the fuselage also the distances of the pylon from the fuselage as well as the impact on the created grids.

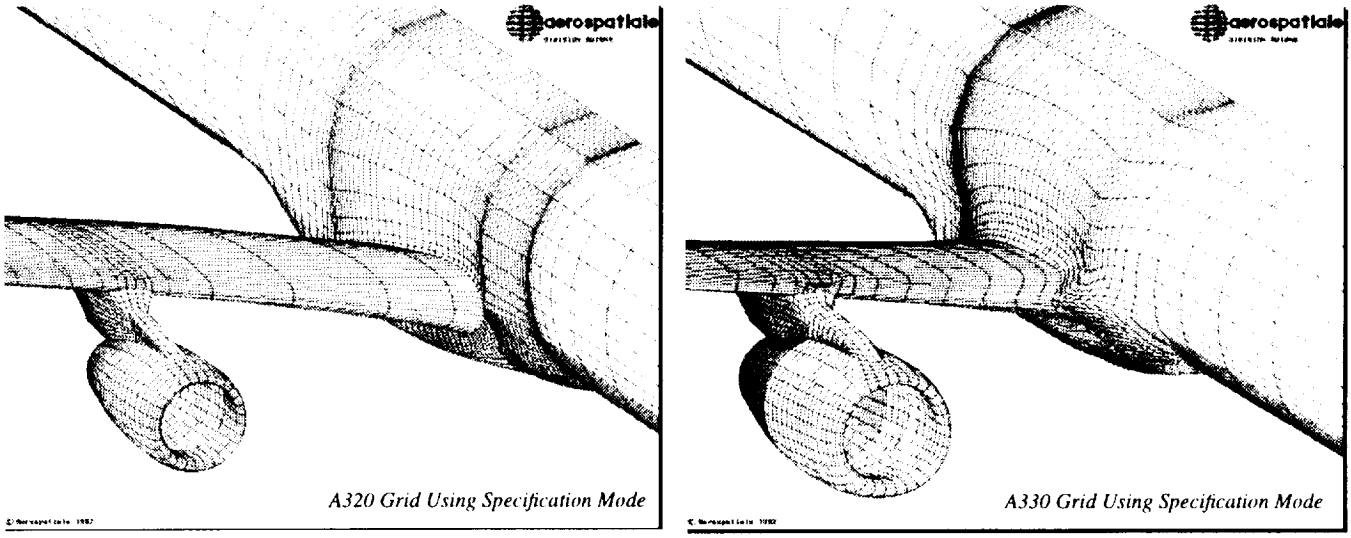



*Figure 3: Using ICEM COMAK, computational volume grid for A320 and automatic generation of grid for the A330 (Courtesy of Aerospatiale).*

## SEMI-AUTOMATIC GRID GENERATION USING ICEM HEXA

ICEM HEXA is an object based semi-automated hexahedral volume mesher for creating multi-block structured meshes or unstructured hexahedral meshes.

The user can define the initial block structure or, alternatively, HEXA will automatically initialize the block structure around a given geometry. Blocks can be interactively adjusted to the underlying CAD geometry. Body fitted internal or external O-Grids can be generated by the system automatically. Mesh sizes can be defined on the object surfaces or individually on the edges using edge meshing options. The grid is projected onto the underlying CAD geometry with minimum user interaction, with complete independence from the orientation of the patches and patch boundaries of the underlying CAD geometry.

Input to ICEM HEXA is CAD geometry, in the form of NURB surfaces, trimmed NURB surface and NURB curves. CAD geometry is either created using ICEM CFD's CAD tool or translated from any other CAD system using IGES or other translator formats. Surface meshes in STL (Stereolithography) format or triangular surface meshes in PATRAN format can also be input as a surface representation for mesh generation.

The following is an example of the grid generation process for a generic chemical processing tank as shown in Figure 4. This configuration contains an inlet, an exhaust port, another cylindrical port for chemical control and two petruding cylinders to the tank. These cylinders are utilized for inspection and controlling the chemical process taking place inside the tank. Each of these cylinders are connected to a smaller cylinders which they connect to pressure regulators. Since this configuration contains many cylinder T-connections, it produces a moderately complex block decomposition strategy for any grid generation system.

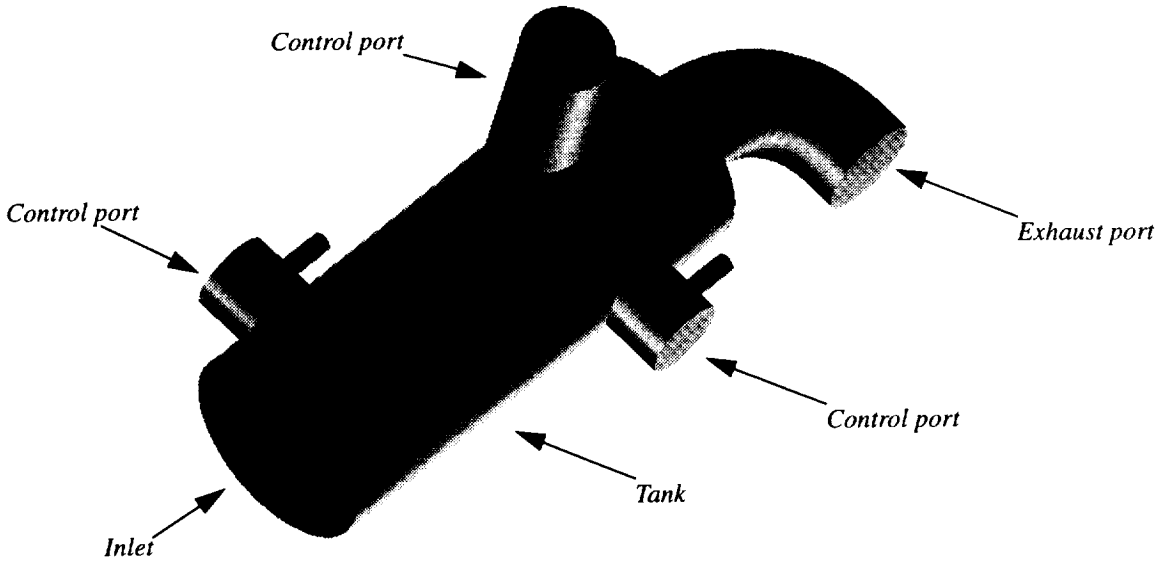



*Figure 4: The chemical processing tank*

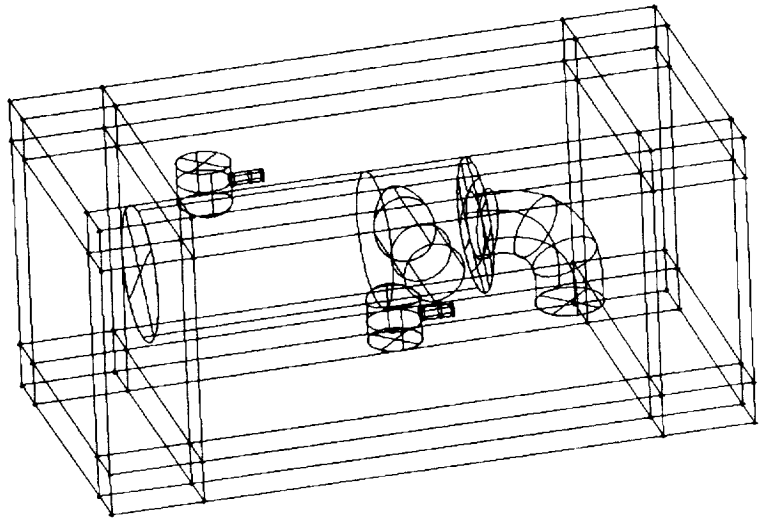


*Figure 5: Block decomposition strategy*

The system automatically initializes the blocking as seen in Figure 5.

Next the blocks are split interactively. During this process the blocks that will be used for grid generation are also selected. After splitting of blocks, the block edges and vertices are fit to the geometry with interactive manipulation. The edges of the blocks can be associated to the curves taken from the CAD model. After the association is done, vertices are moved onto the curves. The blocking structure is shown in Figure 6. After the initial fit, blocks are split to provide control over the critical areas such as the control ports.

After the initial fit, blocks are split further to provide control over the areas such as the pressure ports. The system starts to build internal index control for the blocks as the split operation takes place. Block indices can be used to control which part of the blocking are visible. Additionally, only those parts of the blocking which are currently visible are split during the splitting operations.

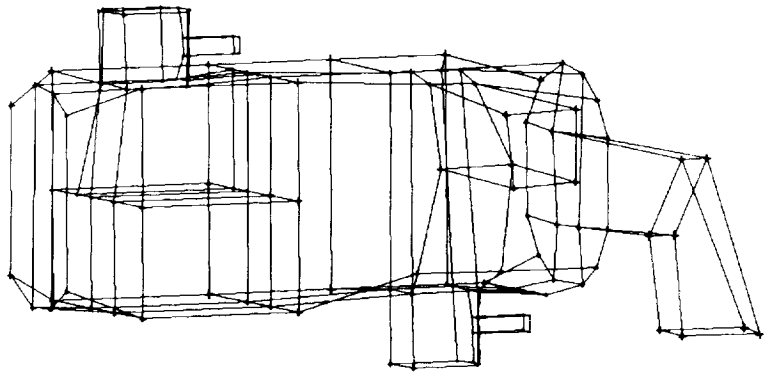


*Figure 6: Block edges of the inner block is being fit to the curves.*

Figure 7 illustrates the blocking structure created. This is only one of many possible blocking strategies. The user may select a strategy that is most appropriate for the type of analysis to be performed. Also seen in Figure 7 is the block edges are fit to the CAD surface geometry.

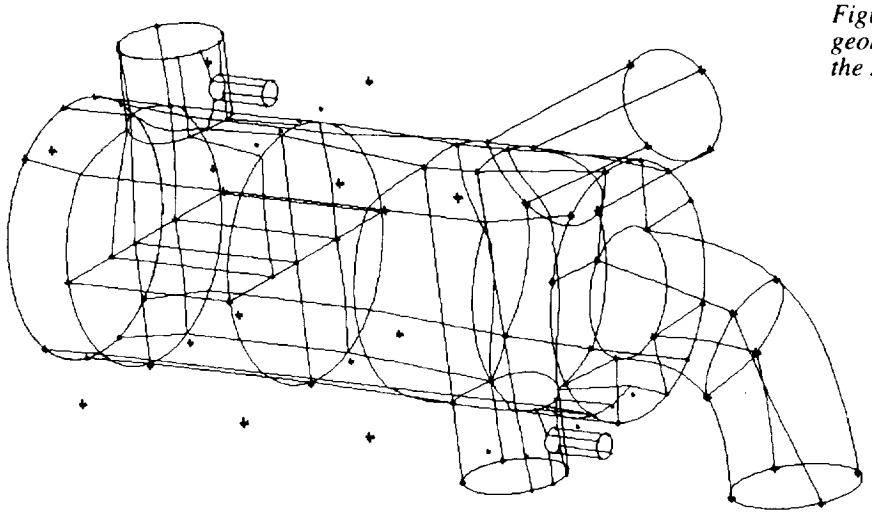


*Figure 7: Blocking of the entire geometry. The block edges are fit to the surface geometry automatically.*

The next step is to define the sizes of the grids on the surfaces of the geometry. The maximum length, initial height and the height ratio off the surface are defined on the object surfaces. If it is necessary, the sizes can later be adjusted on the edges individually.

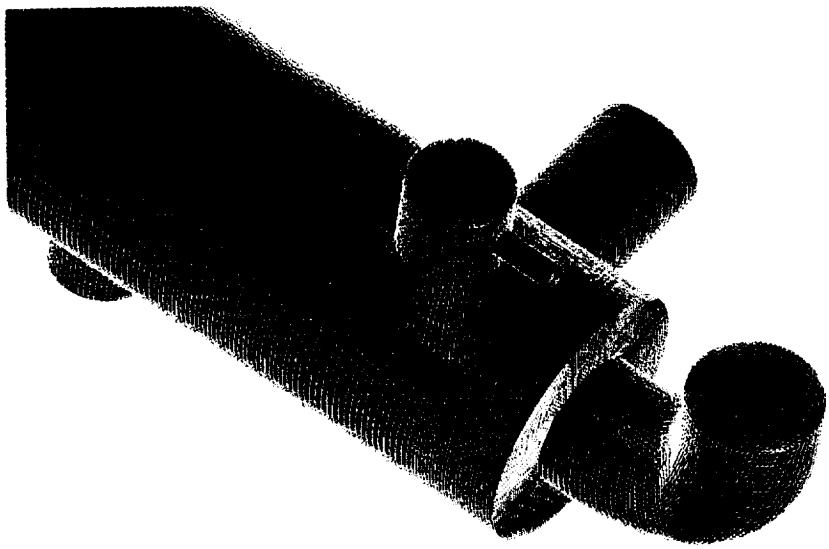


*Figure 8: O-Grid generation around the sail and the rudders*

Figure 8 shows the initial calculated grid. This grid contains many skewed cells since every cylinder is fitted with a single block. To improve the grid quality O-grid generation is needed. Built in tools allow the automatic O-Grid creation. First the blocks are selected and then the faces of these blocks that the O-Grid should pass through. Using the scale factor parameter the distance between the internal block to the walls of the external block is specified.

We will illustrate this feature on one of the control ports. Figure 9 shows the resultant blocking structure after the O-grid is created. The interface of the blocks between the pressure cylinder and the control cylinder are generated automatically by the system..

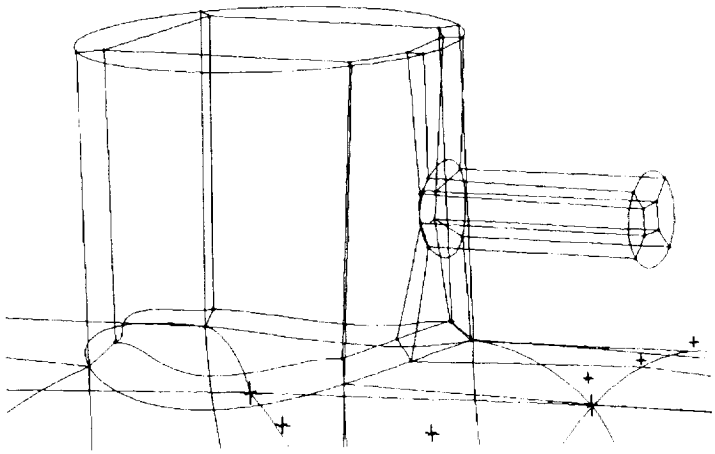


*Figure 9: Surface and the cross sectional volume grid.*

Figure 10 shows the computational grid around the control port with O-Grids. Using this approach the resultant computational grid skewness is 60% and above. The skewness on the perfect rectangular brick is measured to be 100%.

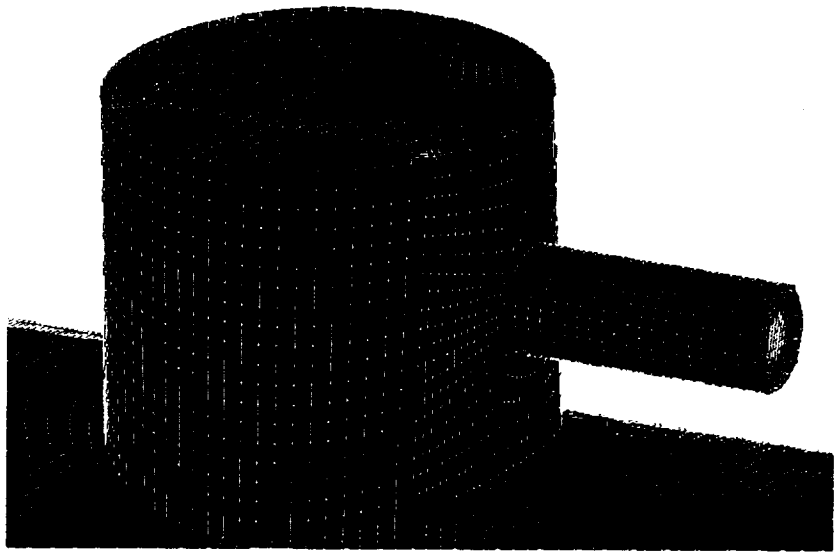


*Figure 10: Automatically created O-Grid around the control port.*

## THE FULLY AUTOMATIC OBJECT BASED GRID GENERATION USING ICEPAK

ICEPAK is a CFD application for thermal management of electronic enclosures. This very easy to use object based package combines model (geometry) generation, automatic grid generation, flow solution with coupled thermal-flow simulation, and post processing into a single environment. It will help the designer to reduce enclosure sizes, eliminate the hot spots, meet noise considerations, increase component density into smaller areas and optimize the locations of fans and vents.

Once the model has been defined using objects, the computational grid is generated automatically. The user either defines the sizes of the elements for each objects, or selects to have ICEPAK calculate the grid automatically based on the objects. A multi-block hexahedral grid is calculated and is body fitted with o-grids around most objects. ICEPAK allows generation of grids at varying levels of complexities. The rules governing each object results in a prioritized O-grid methodology whereby each object is meshed individually as tightly as the user specifications permit in order to resolve the physics of the solution. Objects like blocks, cylinders, thin inclined walls, and wedges can be considered. The following example illustrates the automatic grid generation capability:

As shown in Figure 11, the cabinet is housing a vent, an opening, a box representing a disk drive, 2 fans, one stack of vertical PCB's (4 in a stack), and a power unit. The generation of the geometry is done by selecting objects and bringing them into the cabinet and placing them using the mouse or entering the coordinates from the key board. Using the mouse one can also resize the objects interactively. As the geometry is being build, ICEPAK compiles the boundary conditions on each object. ICEPAK also monitors the

information on each object for grid generation. Each Icepak object has a set of rules or parameters associated with it which are used to guide the generation of the grid around the object. User can selectively modify any individual parameter for any object and then regenerate the computational grid automatically. Any parameter specifically modified and toggled on will be enforced during the grid generation process. This procedure is used to

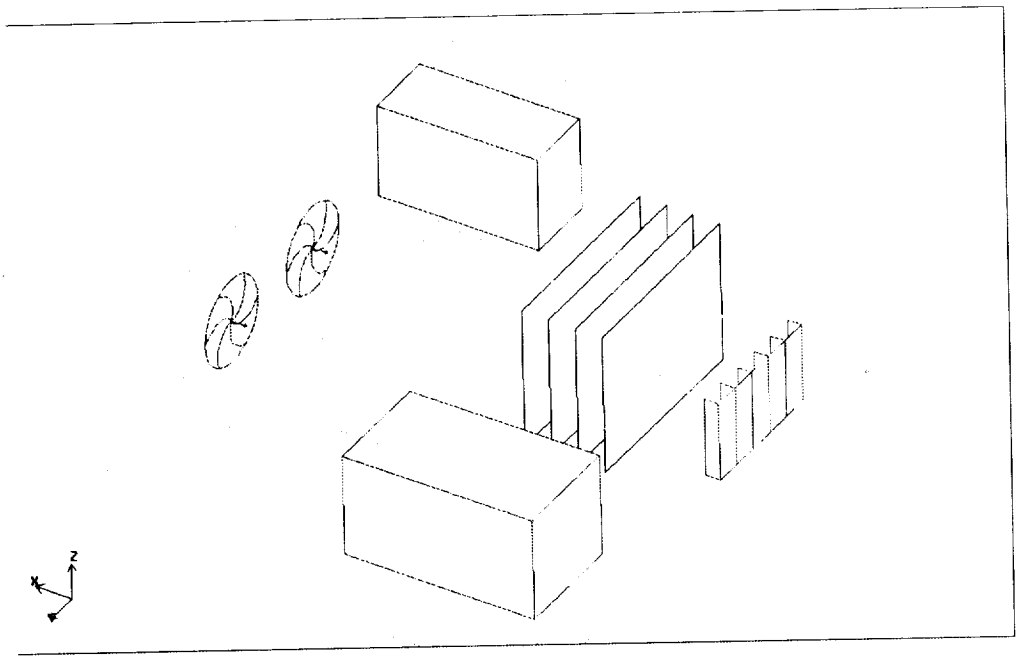



*Figure 11: Cabinet housing PCB's, a disk drive, a power unit, 2 fans, vent and a opening.*

selectively refine the grid around a particular object.

Shown in Figure 12 and Figure 13 are the result of the automatic grid generation process. The plane goes through the middle of the cabinet where the griding of the disk drive and the fans result in O-type grids. General grid generation guideline is to use the minimum count option in conjunction with the maximum x, y, z size option and possibly the maximum initial height option. The maximum x, y, z size option limits the maximum length of any grid element in the corresponding x, y or z coordinate direction.The maximum unit height specifies the maximum height of the first element layer generated around any object (PCB, block, fan, etc.) in the grid. Surface grids on individual objects can be displayed. Thus, for example, you could study the grid on the surface of all blocks in the model or restrict the view to a single specific block. It is also possible to view the surface grid on all objects simultaneously or display the entire mesh

Fully automated grid generation is made simple and fast with ICEPAK. User can generate very complicated grids in minutes rather then weeks. Since the knowledge and the decision making of grid generation process is build into the software, the user can concentrate on other aspects of the analysis such as if objects have been put in the correct place, the solution costs and post processing of computed results.

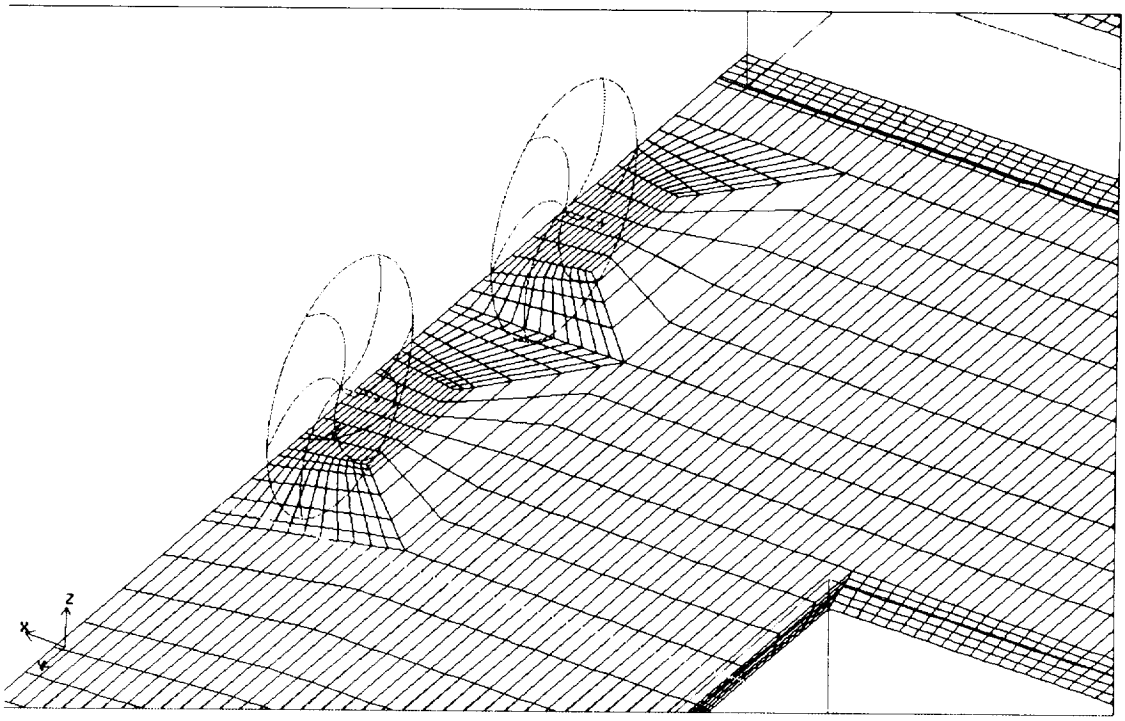


*Figure 12: Automatically generated O-Grid around the fans*

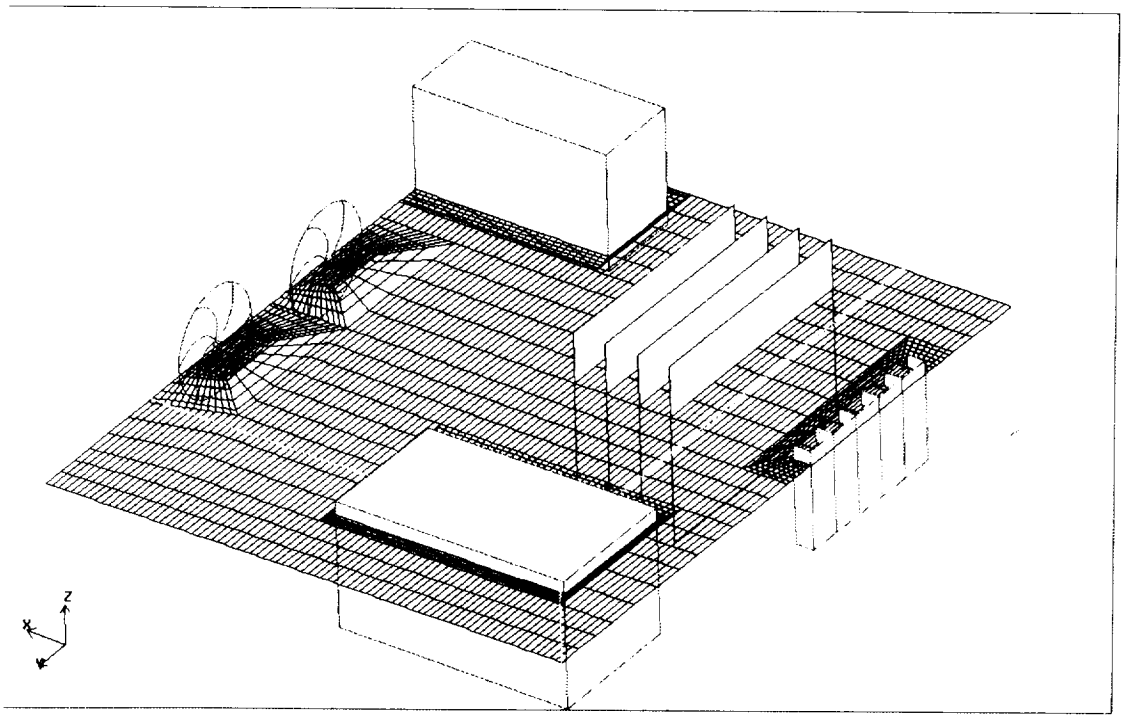


*Figure 13: Fully automatic grid generation inside the cabinet housing*

## CONCLUSIONS

Given the current trend towards more accurate and complete representations of complex flowfield, it is important to have enhanced tools suitable for geometry modeling and grid generation. The current interactive approach for geometry modeling and grid generation allows the direct decision making needed to handle the wide variety of geometries possible. But the expense of the time of the application engineer is needed to be reduced further by providing smart tools for grid generation. The added tools to the current ICEM CFD, such as ICEM COMAK for parametric grid generation and ICEM HEXA for rapid grid generation; will shorten the time for computational grid generation significantly.

The object oriented grid generation tool as implemented in ICEPAK is definitely the new trend in computational grid generation. It is necessary to extend these capabilities to cover a wide variety of shapes.

## REFERENCES


1. Integrated Geometry and Grid Generation System for Complex Configurations, Surface Modeling and Grid Generation Symposium, NASA Langley Virgina, 1992

2. ICEPAK Reference Manual, June 1994, Revision 1.0, 1st Edition. Fluid Dynamics International, Inc.

3. Tcl and the Tk Toolkit, John K. Ousterhout, Addoson-Wesley Publishing Company, July 1994

4. A new Automatic Grid Generation Environment for CFD Applications, D. Bertin, C. Castles, J. Lordon, AIAA CFD Applications Meeting, June 1994, Stanford, CA U.S.A.

# SURFACE GRID/GEOMETRIC GRID GENERATION

N95- 28753

# Geometry modeling and grid generation using 3D NURBS control volume*

Tzu-Yi Yu †
Bharat K. Soni ‡
Ming-Hsin Shih ⋆

Engineering Research Center for Computational Field Simulation
Mississippi State University, MS 39762

## ABSTRACT

The algorithms for volume grid generation using NURBS geometric representation are presented. The parameterization algorithm is enhanced to yield a desired physical distribution on the curve, surface and volume. This approach bridges the gap between CAD surface / volume definition and surface / volume grid generation. Computational examples associated with practical configurations have shown the utilization of these algorithms.

## INTRODUCTION

Surface grid generation is the most labor intensive part of the overall complex three dimensional grid generation process. Also, a significant amount of effort is required in changing the resolutions (grid sizes) and / or the distribution of the grid while maintaining geometry fidelity. In the last few years, various researchers have concentrated on utilizing the Computer Aided Geometry Design (CAGD) techniques [Ref 2,13,14] to expedite the overall surface generation process.

There are many approaches for representing sculptured geometry, such as rational or non-rational Bezier, parametric form, rational or non-rational B-spines, ... ,etc. Among these representation, the Non-Uniform Rational B-Splines (NURBS) has been widely accepted among these researchers. NURBS has been widely utilized to represent and design geometry in the CAD/CAM and the graphics community due to its powerful features, such as the local control property, variation diminishing, convex hull and affine invariance [Ref 1,2]. Also the geometry tool kits, such as curve/surface interpolation, data reduction, degree elevation, knot insertion and splitting, are well-developed [Ref 2,3,4]. These properties have made NURBS representation very popular in recent developments in CAD/CAM and hence in grid generation. For example, in 1992, the scientists of NASA research centers formed a NASA IGES (Initial Graphic Exchange Specification) committee and defined a format named NINO (NASA IGES NURBS ONLY) standard to encourage the CFD community to follow this standard with the NURBS definition for the geometry communication between systems [Ref 5].

The development of the software based on NURBS representation package: CAGI (Computer Aided Grid Interface) was initiated by authors [Ref 15] under the sponsorship of NASA Marshall Space Flight Cen-

* This work is sponsored by NASA Marshall Space Flight Center.

† Graduate Student, AIAA Member
‡ Professor of Aerospace Engineering, AIAA Member.
⋆ Post-Doctor Fellow, AIAA Member.



PRECEDING PAGE BLANK NOT FILMED

ter. The purpose of this paper is to present the progress realized in enhancing the NURBS based curve / surface grid generation techniques into a 3D volume grid generation technique. To this end, various options for generating 3D volume geometry-grid are discussed. A reparameterization scheme has been developed to achieve desired distribution in physical space. Computational examples for modeling practical configurations have been exercised using the volume options and the reparameterization scheme.

## NURBS FORMULATION

Detailed mathematical discussion on NURBS can be found in [Ref 2,3,4]. A brief definition of NURBS curve / surface / volume is presented as follows:

A NURBS Curve of order $k$ is defined as :

$$C(t) = \frac{\sum_{i=0}^{n} W_i \, d_i \, N_i^k(t)}{\sum_{i=0}^{n} W_i \, N_i^k(t)} \qquad (1)$$

where the $d_i$ $i=0,..,n$ denotes the deBoor control polygon and the $W_i$ are the weights associated with each control point. $N_i^k(t)$ is the normalized B-spline basis function of order $k$ and is defined over a knot vector $T=T_i$ $i=0,..,n+k$ by the recurrence relations as shown in equation (2).

$$N_i^k(t) = \frac{(t - T_i)N_i^{k-1}(t)}{T_{i+k-1} - T_i} + \frac{(T_{i+k} - t)N_{i+1}^{k-1}(t)}{T_{i+k} - T_{i+1}} \qquad (2)$$

$$N_i^1(t) \begin{cases} = 1 & \text{if } T_i \le t < T_{i+1} \\ = 0 & \text{otherwise} \end{cases}$$

Throughout this paper, it is assumed that the knot vector has the form $T=\{0, \ldots, 0, T_k, \ldots, T_n, 1, \ldots, 1\}$ with the multiplicity $k$ for the knot value $0$ and $1$ on both ends of the knot vector. If the knot vectors do not match this format, the knot insertion [Ref 6,7] technique must be used to achieve the multiplicity of $k$ on the ends of the knot vector, and if the end knot values are not 0 and 1, the knot vector must be normalized by the last knot value to match this format.

The NURBS surface is the extension of the curve from 1D to the 2D tensor product parametric surface and is shown as equation (3).

$$S(s,t) = \frac{\sum_{i=0}^{m}\sum_{j=0}^{n} W_{ij} \, d_{ij} \, N_i^{k1}(s) \, N_j^{k2}(t)}{\sum_{i=0}^{m}\sum_{j=0}^{n} W_{ij} \, N_i^{k1}(s) \, N_j^{k2}(t)} \qquad (3)$$

Where the $d_{i,j}$ denotes the 3D control net and the $W_{i,j}$ are the weights associated with each control points. $N_i^{k1}(s)$, $N_j^{k2}(t)$ denote the normalized B-Spline basis functions of order $k1$ and $k2$ in the $I$ and $J$ directions, respectively.

The formula for 3D NURBS volume is defined analogous to NURBS surface and is a 3D tensor product form written as equation (4).

$$V(s,t,u) = \frac{\sum_{i=0}^{m}\sum_{j=0}^{n}\sum_{l=0}^{p} W_{ijl}\, d_{ijl}\, N_i^{k1}(s)\, N_j^{k2}(t)\, N_l^{k3}(u)}{\sum_{i=0}^{m}\sum_{j=0}^{n}\sum_{l=0}^{p} W_{ijl}\, N_i^{k1}(s)\, N_j^{k2}(t)\, N_l^{k3}(u)} \qquad (4)$$

The $d_{ijl}$ form the 3D control volume and the $W_{ijk}$ are the weights associated with each control points. And $N_i^{k1}(s)$, $N_j^{k2}(t)$ and $N_l^{k3}(u)$ are the normalized B-spline basis functions of order *k1*, *k2* and *k3* in the *I*, *J* and *L* directions (Instead of using *I, J, K*, this paper uses *I, J, L* to denote the three directions), respectively.

## 3D NURBS CONTROL SURFACE

NURBS has been used to model geometry in CAD/CAM for a long time. Manipulation tools such as "knot insertion" and "degree elevation" are used to increase the flexibility of manipulating the entire geometry. The famous algorithms for knot insertion are "*Boehm*" algorithm [Ref 6] and "*Oslo*" algorithm [Ref 7]. Both knot insertion and degree elevation are fundamental and powerful tools and are very frequently used in modeling the NURBS geometry. The curve and surface interpolation [Ref 8,11] techniques are also used to transform the discrete geometry to B-spline definition. These algorithms have been used as basic tools in modeling the NURBS geometry. Also, it has been shown [Ref 8,9] that many geometries can be represented analytically with a very concise control polygon (control net). For example, a NURBS circular arc can be defined with only 3 control points. Instead of storing the surface grid points, one can store the associated control polygon (control net or volume for the surface and the volume) with the associated weights to reduce the memory load, and this has been considered as one of the features of the NURBS representation.

The algorithms to construct NURBS representation for ruled surface, extruded surface, surface of revolution ...,etc. along with analytical geometries with appropriate data reduction algorithm are well developed and well documented in the CAGD literature. These algorithms have been enhanced and incorporated in the CAGI system [Ref 15]. The application of these algorithms to practical CFD related grid configurations is demonstrated in Figures 1~4. Figure 1 shows four NURBS control patches and their surfaces. This multiple-duct engine is created first by reading the curve profiles, interpolating these discrete data set to NURBS curves, then performing the data reduction [Ref 10] to reduce the redundant control points, and then finally, by using the NURBS revolution [Ref 3,4,13] algorithm. Similar procedures applied to create the single rotation propfan shown in Figure 4. The teapot shown in Figure 2 is modeled by three NURBS control nets, while the missile configuration with four fins shown in Figure 3 is generated by the eight control nets.

## 3D NURBS CONTROL VOLUME

The widely used technique to algebraically generate the three dimensional volume grid is by utilizing the transfinite interpolation algorithm based on the bounding surface grids. The ultimate objective of the present research effort is to explore various NURBS control volume options applicable to three dimensional grid generation. In this paper progress realized in the development of ruled NURBS volume, extruded volume, volume of revolution and composite volume are discussed.

Ruled NURBS volume.

The easiest way to form a 3D NURBS volume is the ruled NURBS volume. The algorithm is described as followed: Given two NURBS surfaces, the first step to form a ruled volume is making the knot vectors of the surfaces be in the same range of [0~1]. Next, considering the *I* direction of both surfaces, use the degree raising technique to raise the low degree of the surface. This procedure will yield a new knot vector and new control net. If the new knot vector differs from the other knot vector, then perform knot

insertion algorithm to merge them into one final knot vector. Then these entire procedures must be applied to the *J* direction of both surfaces again. After this step, the two NURBS surface will have the same orders and the same knot vectors in both *I* and *J* directions. This means the resolutions of control net of both surfaces will be the same. Therefore, one can connect the corresponding control point together to form the 3D NURBS volume. In other words, the orders and knot vectors of the final volume in *I* and *J* directions will be the same as those of the surfaces after degree elevation and knot insertion, and the order in *L* direction will be set as two with knot vector set as (0,0,1,1). Figure 5 shows a 3D "apple-like" NURBS volume formed by this algorithm.

While defining the NURBS ruled surface, the IGES defines a variable named "*DIRFLG*" as a direction flag to control the direction of how the surface will be connected. If DIRFLG is 0, then the surface will connect the points on the same direction of the two curves, otherwise, the direction of one of the curves will be reversed. Similar to this definition, it is possible to set two flags as "*DIRFLG_I*" and "*DIRFLG_J*" to control how the directions of the two surfaces will be connected. This increases the flexibility of generation options.

Extruded volume.

IGES defines the extruded surface as a surface formed by moving a line segment (called generatrix) parallel to itself along a curve(called directrix). In other words, given a NURBS curve, one can generate another curve by extruding the given curve with a distance $\alpha$ along a vector *V*. Similar to this definition, the NURBS extruded volume is defined as: given a NURBS surface, a new surface can be generated by extruding the given surface with a distance $\alpha$ along a vector *V*. Mathematically, this new extruded surface can be described as $\vec{d}_{ij} = d_{ij} + \alpha\vec{V}$ with the same orders, same knot vectors as those of the given surface. After this step, the algorithm of "ruled volume" can be applied to these surfaces to form a final NURBS volume. Figure 6 shows a 3D NURBS extruded volume.

Volume of revolution.

IGES also defines the surface of revolution entity as the surface which is formed by rotating a given curve (called generatrix) with respect to an arbitrary straight line (axis of revolution) from a starting angle ( not necessarily zero) to an ending angle. Compared to most of the literature which discussed the full revolution (rotation angle is set to $360^0$), this definition is more general and useful in grid generation. The NURBS volume of revolution discussed in this paper is the extension of the surface of revolution, and it is formed by rotating a given NURBS surface with respect to an arbitrary line as the axis of revolution from a starting angle to an ending angle. This general algorithm can be stated as follows: First step is translating / rotating the axis of revolution by proper transformation matrix so that it is coincident with the Z axis. Also, apply this transformation matrix to the given NURBS surface so that the entire surface can be kept in the same position as the axis of rotation. It is assumed that the surface is defined as NURBS with the control net $d_{ij}$, order *k1* and *k2*, weights $W_{ij}$ and two knot vectors. Next, for each control net $d_{ij}$ (on the generatrix $i= 0,..,m$, $j= 0,..,n$), construct the control volume $d_{ijl}$ $l=0$ .. $p$ at each $j$-th cross section according to the starting and ending angle by utilizing the circular arc algorithm. In other word, this approach constructs the NURBS control net at each $j$ constant plane by revolving the control polygon $d_{iJ}$ with respect to *L* direction and then "stacks" them together to form a final NURBS volume. Figure 7 demonstrates this approach. The procedure for generating the NURBS circular arc can be referenced in [Ref 9,13] and the $p$ (for the last dimension of control volume) is determined by the sector angle (equal to the difference between ending and starting angle). For example, if the angle is less than $90^0$, $p$ is equal to 2. If the angle is in the range of $90^0$ ~$180^0$, $p$ is equal to 4, if in the range of $180^0$ ~$270^0$, $p$ is 6, if it is greater than $270^0$, $p$ should be 8. For the section angle $\theta$, the weights are set as (in each $J$ constant plane, $J=0,..n$) $W_{iJp} = w_{iJ}$, $w_{iJ}\ cos(\theta/p)$, $w_{iJ}$, $w_{iJ}cos(\theta/p)$ ,... $i=0,..,m$ ( repeat $w_{iJ}$, $w_{iJ}cos(\theta/p)$ with total $p$+1 terms). The knot vectors in directions of *I* (*s*) and *J* (*t*) are the same as the ones of the given surface while the knot vector in direction *L* (*u*) is determined according to the circular arc procedure. For example, when $p$ = 2, the associated knot vector is set as (0,0,0,1,1,1), for the case of $p$ = 4, the knot vector is set as ( 0, 0, 0, 1/2, 1/2, 1, 1, 1), for the case of $p$ = 6, the knot vector is set as (0, 0, 0, 1/3, 1/3, 2/3, 2/3, 1,1,1) and the last case when $p$ = 8, the knot vector is (0,

0, 0, .25, .25, .5, .5, .75 , .75 ,1,1,1). Also, set the orders in $I$ and $J$ be $k1$ and $k2$ (as the ones in original surface) while set 3 as the order in $L$ direction. Since the NURBS has the translate and rotate invariant property, one can apply the inverse transformation matrix to the control volume (without altering the weights and knot vectors) back to the original coordinates. Figure 8 shows a 3D NURBS control volume developed by this approach.

Several complex circular pipes can be constructed by using the combination of the aforementioned approaches. Figure 9 (a) and (b) are formed by first building a NURBS surface, then extruding it with a desired distance to form a cylinder pipe, extracting the last cross sectional surface from this cylinder pipe and then performing the volume of revolution algorithm to create those turning portions, and again extracting the last cross sectional surface and using the ruled surface algorithm to form the last pipe. Last, the composite method should be performed to construct the entire volume pipe.

Composite volume.

A composite NURBS volume is defined as a volume consisting of lists of constituent volumes. The compositing procedure is stated as follows: Suppose two constituent NURBS volume $V_1$ and $V_2$ form a composite volume. Assume that $V_1$ has control volume $d_1[0{:}m_1, 0{:}n_1, 0{:}l_1]$, weight $W_1[0{:}m_1, 0{:}n_1, 0{:}l_1]$, three knot vectors $knot_i)_1$, $knot_j)_1$, $knot_l)_1$ and orders $k_i)_1$, $k_j)_1$, $k_l)_1$ while $V_2$ has control volume $d_2[0{:}m_2, 0{:}n_2, 0{:}l_2]$, weight $W_2[0{:}m_2, 0{:}n_2, 0{:}l_2]$, three knot vectors $knot_i)_2$, $knot_j)_2$, $knot_l)_2$ and orders $k_i)_2$, $k_j)_2$, $k_l)_2$. There are many combinations when two volumes are joined together. For example, one may join the volumes in $I$ direction with the interface of $J, L$ surface, or join in $L$ direction with the interface of $I, J$ surface, .. etc. Even though there are many cases, the procedure is similar. Take the case when joining in $I$ direction as an example, the first step is to perform the degree elevation to $V_1$ and $V_2$ so that these two volumes can have the compatible degrees in $I, J$ and $L$ directions. If the two knot vectors in $J$ direction for $V_1$ and $V_2$ are not the same, merging them together by setting the final knot vector as $\{ knot_j)_1 \cup knot_j)_2 \}$, then applying the knot insertion to $V_1$ and $V_2$ in $J$ dimension. Same procedure should be applied to $L$ direction if $knot_l)_1$ and $knot_l)_2$ are not the same. After this step, $V_1$ and $V_2$ will have the same degree in three directions, and the number of control points and knot vectors in $J$ and $L$ directions will be the same. The second step is to adjust the knot vector $knot_i)_2$ so that its first knot value can be the same as the last knot value $knot_i)_1$. Shifting the knot vector will not change the original NURBS because the basis function is a "normalized" basis function. The third step is to build up the final knot vector by joining the two knot vectors into one knot vector and set that knot value at the joint point to have the multiplicity equal to ($order$ -1). For example, if the knot vector $knot_i)_1$ is [0, 0, 0, 1, 1, 1] and the knot vector $knot_i)_2$ is [2,2,2,3,3,3], adjust the second knot vector by shifting -1 to each value. Thus, the $knot_i)_2$ becomes [1,1,1,2,2,2]. Suppose the final $order$ of these two volumes in $I$ direction is three, then, the final knot vector should be [0,0,0,1,1,2,2,2] (one may notice the interior knot 1 has multiplicity of ($order$-1)=2). The fourth step is to match the weights. This step can be illustrated as the following pseudo code.

```
for (L=0; L<=l1; L++)
for (j=0; j<=n1; j++)
{
   let factor = W1[m1, j, L] / W2[0, j, L] ;

   for (i=0; i<=m2; i++)
     W2[i, j, L] = W2[i, j, L] *factor ;
}
```

The last step is to build up the final control volume and weights by throwing the $d_2[0{:}0, 0{:}n_2, 0{:}l_2]$ and $W_2[0{:}0, 0{:}n_2, 0{:}l_2]$ away and jointing the others as one control volume and weights.

## REPARAMETERIZATION PROCEDURE

As discussed earlier, it is clear that the NURBS representation is defined in a parametric form. Take a curve as an example: any value $t$ in parametric space will result in a point $C(t)$ in the physical space. This

property leads to some advantages and disadvantages. In perspective of numerical grid generation, the most conspicuous contributions for NURBS are that it provides a very easy and intuitive way for changing topologies, resolutions and the distributions of a surface grid. The surface grids generated in grid generation tools often have different topologies - such as O type, C type and H type grid or even unstructured or hybrid grid. Generating a 3D surface from these different topologies may require repeated applications. However, by using the 3D NURBS control polygon and weights the re-mapping and redistribution can be achieved by re-establishing the associated distribution mesh (network of parametric values). Figure 10~11 demonstrate the examples of the redistribution process. In these example, the structured and unstructured (hybrid) grid is generated just by constructing the associated hybrid distribution space. The other advantages are dealt with the resolution and distribution. Due to this parametric property, evaluating the same NURBS information (same control points, weights, knot vector and the same orders) with different parametric mesh (includes the different dimension of grid sizes) can yield a surface grid redistribution. This is accomplished by the "reparameterization" procedure.

What makes NURBS so popular in CAD design is that it provides many stable geometry manipulating properties. One of those properties is the local control property. The engineer can locally modify the NURBS geometry by changing the NURBS information (such as the location of the control polygon,weights or even knot vectors) without perturbing the entire geometry. However, this reshaping also changes the effect of parametric space on the distribution of points, often resulting in unacceptable distribution. This is due to "bad parameterization"-- which means after changing any of the NURBS information (control polygon, knot vectors or weights), the desired distribution in parametric space will not necessarily result in the desired distribution in physical space (take equation (1) as an example, the evenly distribution in $t$ (parametric space) doesn't result in evenly distribution in $C(t)$ (physical space)). These situations are demonstrated in Figure 12. The curve itself has been locally modified, yet one can also notice that the points on the modified curve are stretched towards the control point for which the weight has been increased. This has been claimed as the disadvantage of the parametric NURBS. Several literatures [Ref 13,14] have shown their approaches to overcome this obstacle and to obtain the desired distribution in the final geometry after any modification. A new and more efficient algorithm for reparameterizating parametric curve has been developed as follows:

Consider a NURBS curve with resolution $ni$. Let

(1) $t_1(i)$, $i$=1,..,$ni$ be the unknown distribution which is used to generate the desired curve in physical space;

(2) $t_2(i)$, $i$=1,..,$ni$ be the normalized chord length of the curve with desired distribution; and

(3) $t_3(i)$, $i$=1,..,$ni$ be the normalized chord length of the curve evaluated at parametric values $t_1(i)$, $i$=1,.., $ni$.

Take Figure 12 as an example, if the designer would like to have the final curve as shown in Figure 12(c), then $t_2(i)$ will be a 1D array which contains the distribution packing towards two ends, and $t_1(i)$ are the parametric values which are to be determined such that the $t_3(i)$, the normalized chord length of final curve, would be the same as $t_2(i)$ (or $|t_2(i) - t_3(i)|$ be minimized for all $i$=1,.., $ni$).

In order to obtain $t_3(i)$, the initial values of $t_1(i)$ must be initialized. One can set $t_1(i)$ as evenly distributed initially, then evaluate the curve with these parametric values to obtain the final curve, then normalize the entire chord length to obtain $t_3(i)$. Generally, the $t_3(i)$ will be different than $t_2(i)$. How to adjust $t_1(i)$ to have new $t_3(i)$ which is close to $t_2(i)$? The entire reparameterization procedure is described in the following computer pseudo code:

```
for (i=1; i<(ni-1); i++)
{
   find the location of j such that t3[j] ≤ t2[i] < t3[j+1].

   let α=t2[i]-t3[j]; β=t3[j+1]- t2[i];
   t[i]=(t1[j+1]*α + t1[j]*β) / (α + β);
}
```

The final $t[i]$ will be the desired distribution curve, and using this distribution to evaluate the NURBS curve shown in Figure 12(b) (with the middle weight increased) will yield the result shown in Figure 12(c), which has the distribution packing towards two ends.

This reparameterization algorithm can also be extended to find the desired distribution mesh for the tensor product NURBS surface. The reasons which lead to the bad parameterization problems for the surface may come from arbitrarily modifying the NURBS information (such as the control point, weights or knot vector), or from significant difference of the arc lengths of two opposite boundaries. These cases are demonstrated in Figure 13 and 14. From equation (3), one can understand what cause the problem shown in Figure 14. There are only two knot vectors which control the distribution in $I$ and $J$ directions. If both of the two opposite boundaries in $I$ (or $J$) have different distributed requirements or have big difference in arc length, the one single knot vector in $I$ (or $J$) may cause problem. Similar to the reparameterization procedure of the curve, the algorithm for the surface is stated as followed.

Instead of defining the three 1D arrays, three sets of 2D arrays $(s_1, t_1)$, $(s_2, t_2)$ and $(s_3, t_3)$ must be defined. Consider a NURBS surface with resolution $ni$ by $nj$. Let

(1) $(s_1(i,j),t_1(i,j))$, $i=1,..,ni$, $j=1,..,nj$ be the unknown distribution mesh which is used to generate the desired distribution of the surface in physical space;

(2) $(s_2(i,j),t_2(i,j))$, $i=1,..,ni$, $j=1,..,nj$ be the normalized chord length of the surface with desired distribution; and

(3) $(s_3(i,j),t_3(i,j))$, $i=1,..,ni$, $j=1,..,nj$ be the normalized chord length of the surface evaluated at parametric values $(s_1(i,j), t_1(i,j))$.

Take Figure 13 as an example. If the designer would like to have the final surface, as shown in Figure 13(c), then $(s_2(i,j),t_2(i,j))$ would be a 2D array which contains the even distribution, and $(s_1(i,j),t_1(i,j))$ would be the parametric values which are to be determined such that the $(s_3(i,j),t_3(i,j))$, the normalized chord length of final surface, would be the same as $(s_2(i,j),t_2(i,j))$ or within certain tolerance.

Similar to curve algorithm, one can set $(s_1(i,j),t_1(i,j))$ as evenly distributed initially, then evaluate the surface with these parametric values to obtain $(s_3(i,j),t_3(i,j))$. Most likely the $(s_3(i,j),t_3(i,j))$ will be different than $(s_2(i,j),t_2(i,j))$. The procedure to adjust $(s_1(i,j),t_1(i,j))$ so that $(s_3(i,j),t_3(i,j))$ can be close to $(s_2(i,j),t_2(i,j))$ is presented with the following computer pseudo code:

```
for (j=1; j<(nj-1); j++)
for (i=1; i<(ni-1); i++)
{
   search the index of I , J such that (s2(i,j),t2(i,j)) is located within the cell of (s3(I, J), t3(I, J)),
   (s3(I+1, J), t3(I+1, J)), (s3(I, J+1), t3(I, J+1)), and (s3(I+1, J+1), t3(I+1, J+1)) ;

   let (s2(i,j),t2(i,j))= (1-α)(1-β)(s3(I, J), t3(I, J)) + (1-α)β(s3(I, J+1),t3(I, J+1)) +
   α(1-β)(s3(I+1, J), t3(I+1, J)) + αβ(s3(I+1, J+1), t3(I+1, J+1)) and solve for α and β ;

   new (s(i,j),t(i,j))= (1-α)(1-β)(s1(I, J), t1(I, J)) + (1-α)β(s1(I, J+1),t1(I, J+1)) +
   α(1-β)(s1(I+1, J), t1(I+1, J)) + αβ(s1(I+1, J+1), t1(I+1, J+1)) ;
}
```

The final $(s(i,j),t(i,j))$ are then the desired distribution mesh in which to generate the desired surface grid shown in Figure 13(c) and 14(d).

The reparameterization algorithm for the volume case is similar. 3 sets of 3D arrays should be used, and instead of bi-linear interpolation, the trilinear interpolation should be used while adjusting the desired distribution.

## CONCLUSIONS

The grid generation algorithm associated with the ruled volume, extruded volume, volume of revolution and the composite volume based on the NURBS representation have been presented and demonstrated

by utilizing practical configurations. The development of the reparameterization scheme and its influence to curve / surface / volume grid distribution is demonstrated by examples. These algorithms have been incorporated as modules in the CAGI system. The research associated with enhancing the volume options and manipulations along with evaluation of the traditional transfinite interpolation algorithm based on NURBS is underway.

## ACKNOWLEDGEMENT

The authors would like give their thanks to Mr. Ted Benjamin and Mr. Robert Williams of NASA Marshall research center for their support of this research project.

## REFERENCE

[1] De Boor, C., - *A Practical Guide to Splines, Applied Mathematical Sciences*, Vol 27, Springer-Verlag,1972.

[2] FARIN, G. "Curves and Surfaces for Computer Aided Geometric Design: A Practical Guide," Third Edition, Academic Press, 1990.

[3] Piegl, L., - *'On NURBS: A survey' IEEE Computer Graphics & Applications*, Vol 11, No 1, pp 57 - 71, January, 1991.

[4] Piegl, L. - *'Rational B-Spline Curves and Surfaces for CAD and Graphics' State of the Art in Computer Graphics, Visualization and Modeling*, Eds, David F. Rogers, Rae A. Earnshaw, Springer-Verlag, pp 225-269, 1991.

[5] Blake M.W.,Kerr P.A., Thorp S.A., and Chou J.J., *"NASA Geometry Data Exchange Specification for Computational Fluid Dynamics (NASA IGES)"*. NASA Reference Publication 1338., April, 1994.

[6] Boehm, W. - *'Inserting New Knots into B-Spline Curves' Computer Aided Design*, Vol. 12, No 4, pp 199~201, July 1980.

[7] Cohen, E., Lyche, T. and Riesenfeld, R.F.,- *'Discrete B-Splines and subdivision techniques in computer-aided geometric design and computer graphics', Comput. Graph. & Image Proc.* Vol. 14, No 2, pp 87~111, 1980.

[8] Tiller, W. - *'Rational B-Splines for Curve and Surface Representation' IEEE Computer Graphics & Applications*, Vol. 3, No 10, pp 61~69, Sep 1983.

[9] Piegl, L. and Tiller, W.- *A Menagerie of Rational B-Splines'* IEEE Computer Graphics & Applications, Vol 9, No 5, Sep 1989, pp 48~56.

[10] Tiller, W.- *'Knot Removal Algorithms for NURBS Curves and Surface' Computer Aided Design*, Vol 24, No 8, pp 445~453. August 1992.

[11] Piegl, L. and Tiller, W.- *'Curve and surface constructions using rational B-splines' Computer Aided Design*, Vol 19, No 9, 1987, pp 485~498.

[12] Boehm W., Farin G. and Kahmann J.- *'A survey of curve and surface methods in CAGD' Computer Aided Geometric Design*, pp 1~60, 1984.

[13] Yu, T.Y. and Soni, B.K.,"Geometry Transformer and NURBS in grid Generation," 4th International Conference on Numerical Grid Generation in CFD and Related Fields, Swansea, UK., April 6-8, 1994.

[14] FARIN, G. "NURB curves and surfaces: from projective geometry to practical use," First Edition, A K Peters, Ltd., 1995.

[15] Yu, T.Y., Soni, B.K., and Shih M. H. "CAGI: Computer Aided Grid Interface," AIAA-95-0243, 33nd Aerospace Sciences Meeting & Exhibit., Reno, Nevada January 09~12,1995.

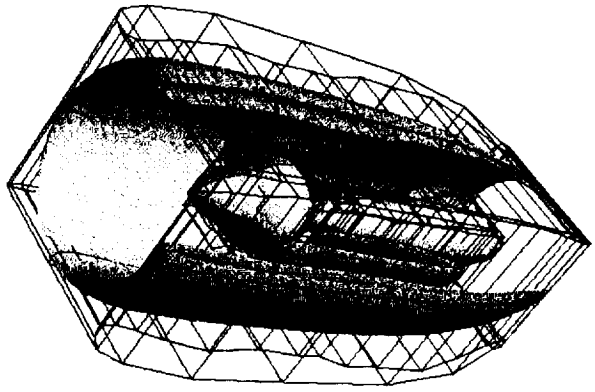


Figure 1. 3D NURBS control nets model the multiple-duct engine.

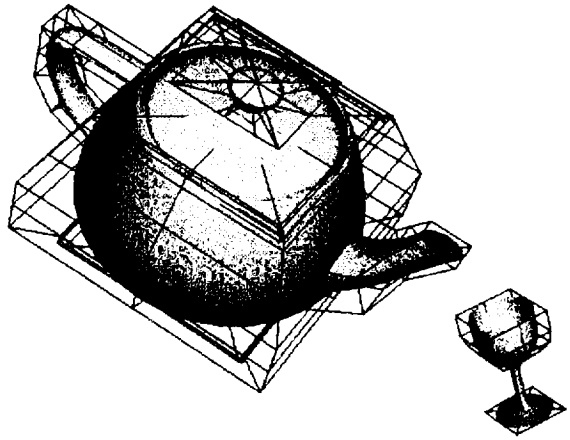


Figure 2. A teaset modeled by 3D NURBS control surfaces.

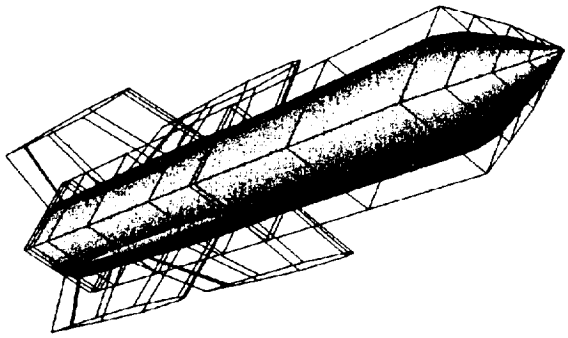


Figure 3. 3D NURBS control patches model the missile (with fins) geometry.

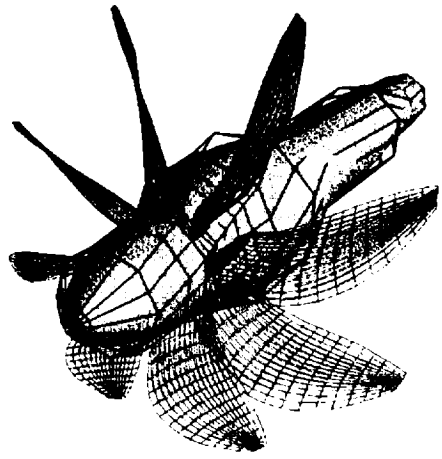


Figure 4. 3D NURBS control patches model the single rotation propfan.

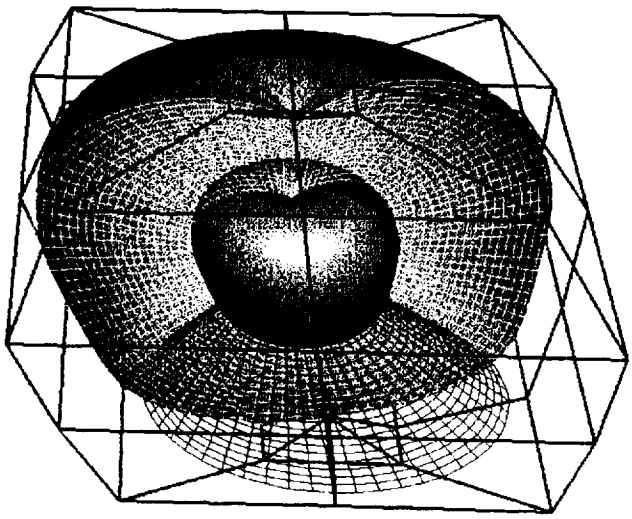


Figure 5. 3D NURBS control volume and its volume grid for ruled volume case.

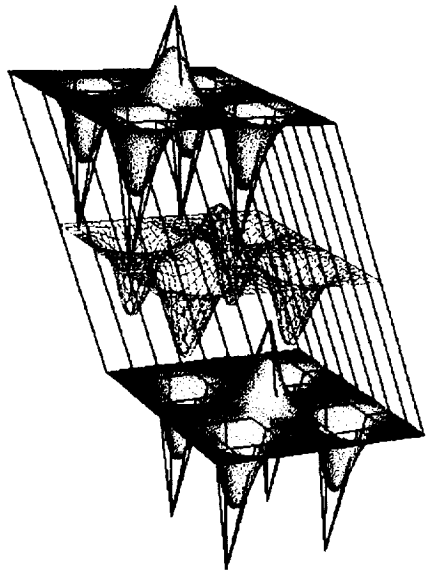


Figure 6. 3D NURBS extruded volume and its volume grid.

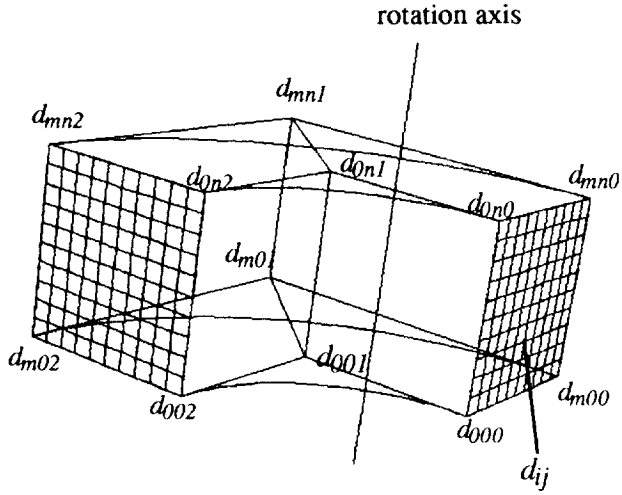



Figure 7. Illustration of volume of revolution.

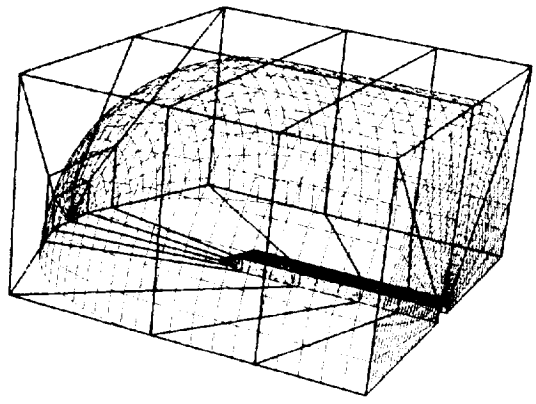
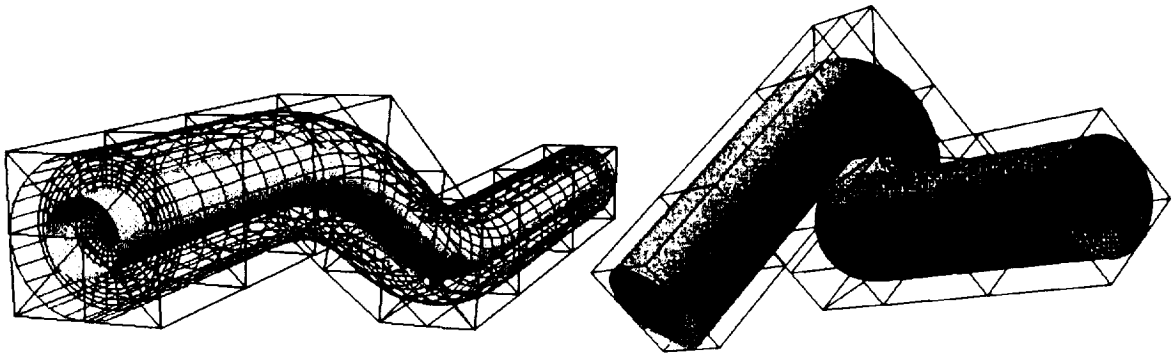
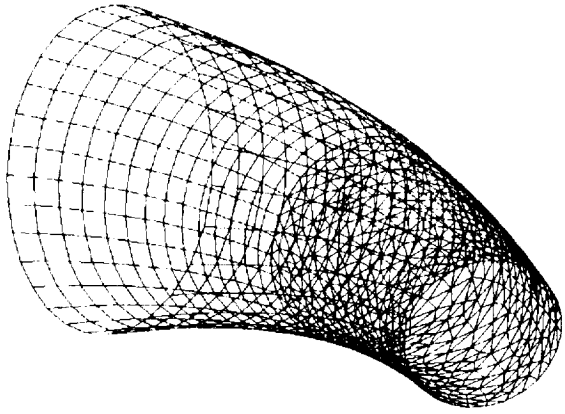


Figure 8. 3D NURBS control volume models the missile configruation.



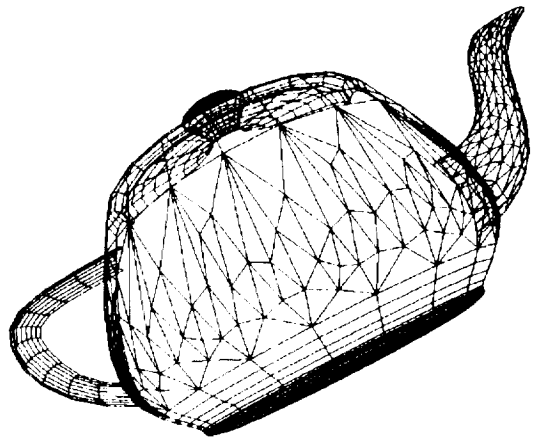
(a) (b)

Figure 9 (a) (b). 3D NURBS circular pipes formed by the algorithm of ruled volume, extruded volume and volume of revolution.



(10)

Figure 10. 3D hybrid pipe.



(11)

Figure 11. 3D hybrid teapot.

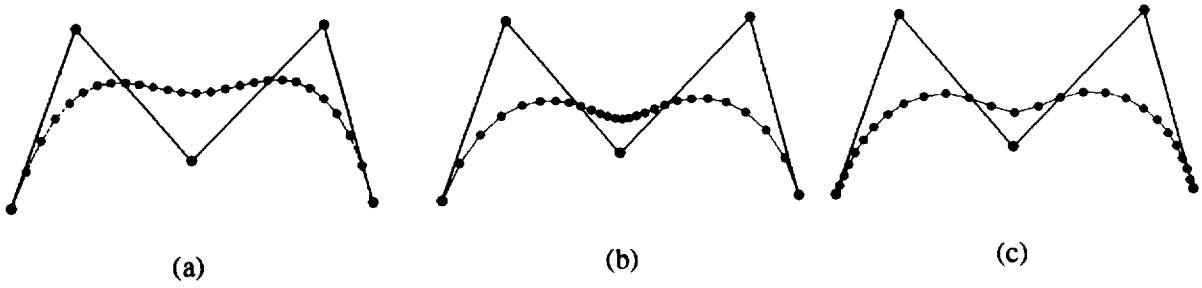



Figure 12. (a) Original NURBS curve and its control polygon.
(b) Increase the weight in the middle control point, evaluated the curve with even distribution in parametric space.
(c) Reparameterization example. The shape of the curve is unchanged, yet with new distribution packing towards two ends.

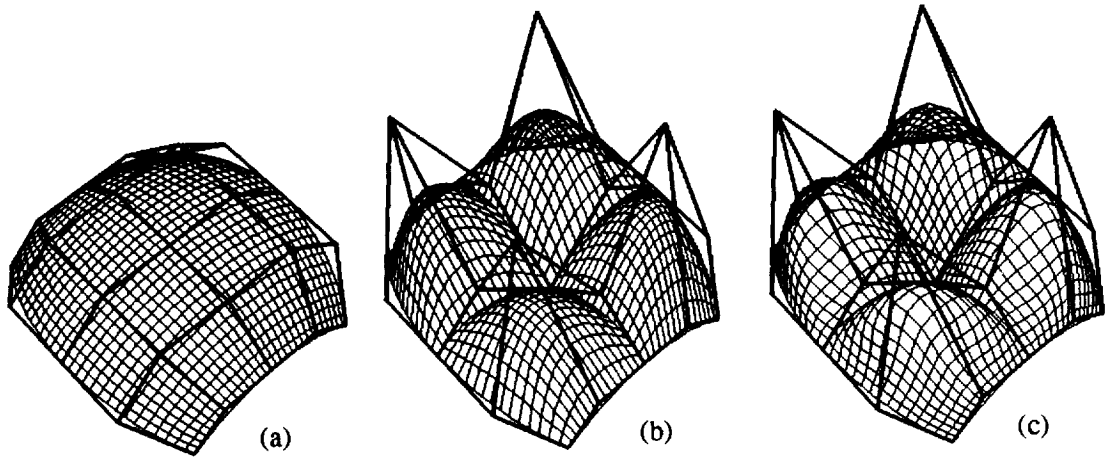
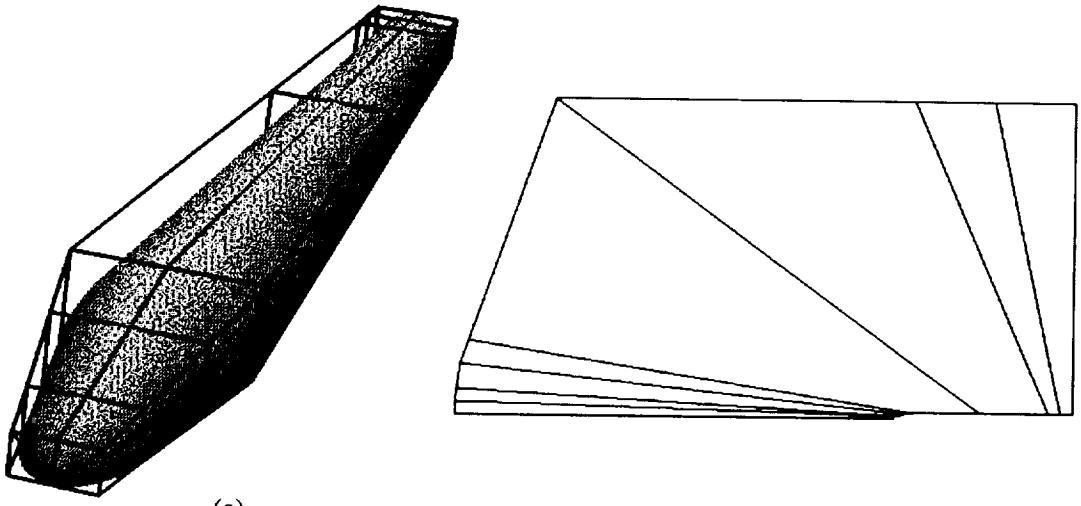


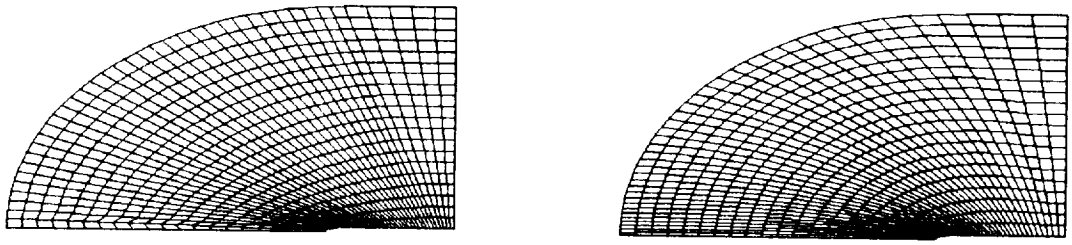

Figure 13. (a) Original NURBS surface and its control net.
(b) Alter the weights and control points location and evaluated the surface with even distribution in parametric space.
(c) Reparameterization example. The shape of the surface is unchanged, yet with new distribution.



(a) (b)

Figure 14 (a). A 3D control net models the missile configuration.

(b). 2D missile control patch for external flow simulation.



(c) (d)

Figure 14 (c). 2D surface grid patch evaluated with even distribution in parametric space.

(d). Reparameterization precedure for a smooth grid.

N95- 28754

# GENERATING GRIDS DIRECTLY ON CAD DATABASE SURFACES USING A PARAMETRIC EVALUATOR APPROACH

**Timothy D. Gatzke* and Thomas G. Melson****
**McDonnell Douglas Corporation**
**St. Louis, Missouri**

## ABSTRACT

A very important, but often overlooked step in grid generation is acquiring a suitable geometry definition of the vehicle to be analyzed. In the past, geometry was usually obtained by generating a number of cross-sections of each component. A number of recent efforts have focussed on non-uniform rational B-spline surfaces (NURBS) to provide a single type of analytic surface to deal with inside the grid generator. This approach has required the development of tools to read other types of surfaces and convert them, either exactly or by approximation, into a NURBS surface. This paper describes a more generic parametric evaluator approach, which does not rely on a particular surface type internal to the grid generation system and is less restrictive in the number of surface types that can be represented exactly. This approach has been implemented in the McDonnell Douglas grid generation system, MACGS, and offers direct access to all types of surfaces from a Unigraphics part file.

## INTRODUCTION

Grid generation tools have progressed rapidly over the past decade with a proliferation of grid generation systems. In order to shorten the time required for grid generation, these tools have focussed on batch[1], interactive[2,3], unstructured grid[4,5,6], and overset grid[7,8] methods. However, all of these tools require a geometry definition in a particular format before grid generation can begin. For the standard test cases such as a NACA 0012 airfoil or the ONERA M6 wing, this geometry is readilly available. Alas, Computational Fluid Dynamics has progressed to the complex configurations of today's and tomorrow's real vehicles and this geometry is primarily generated within Computer Aided Design (CAD) systems.

Several popular CAD systems have developed along parallel paths. Due to the competitiveness of the CAD industry, most of this development has remained proprietary. Coupled with the variety of design needs, this has led to development of a large number of surface types. Attempts to standardize data exchange such as the Initial Graphics Exchange Specification (IGES)[9] and Product Data Exchange Using STEP (PDES)[10] still allow for user defined surface types which are not common to all systems. As a result, today's CAD models are composed of a wide range of surface types, complicating efforts to work directly on the CAD surface types.

In the past, geometry was usually obtained by generating a number of cross-sections of each component. This either involved communicating the CFD engineer's needs to a designer, or

---

* Principal Technical Specialist - Computational Fluid Dynamics Project

** McDonnell Douglas Fellow - Advanced Integrated Mathematical Systems



PRECEDING PAGE BLANK NOT FILMED

teaching the CFD engineer to use the CAD system. The time for this step is dependent on the complexity of the CAD model and could take a few weeks for a full vehicle such as an F/A-18 aircraft. Other efforts have approached the problem of acquiring geometry by converting the initial CAD surfaces to non-uniform rational B-spline surfaces (NURBS)[11] to provide a single type of analytic surface to deal with inside the grid generator. However, for many surface types there is no exact NURBS representation and so the surface must be approximated. This introduces the additional step to convert surfaces to NURBS and the associated development cost for tools to convert or approximate every surface type. This paper describes a more generic parametric evaluator approach which does not rely on a particular surface type internal to the grid generation system and is less restrictive in the number of surface types that can be represented exactly. This approach has been implemented in the McDonnell Douglas grid generation system, MACGS, and offers direct access to all types of surfaces from a Unigraphics part file.

## UNIGRAPHICS AND LOFT DATA

The rapid development of CAD tools over recent years has led to current aircraft defined on a wide range of CAD systems. Over the last three years, McDonnell Douglas has been transitioning to the Unigraphics II CAD sytem. One of the important issues in this transition is how to handle data from aircraft designed earlier on the legacy CAD systems. CAD systems have been proprietary and their internal geometry representation, and the routines to work with geometry have been closely guarded. As a result, each system has its own set of surfaces and formats. A few of these may be common to all systems but a large number are unique to each system.

Another driver towards a large number of surface types is the lofting process. Lofting is the process of defining the external surface of a vehicle. This surface definition is then used for the detail design of the vehicle. Manufacturability is a key concern which requires attention to surface continuity and smoothness along with knowledge of manufacturing processes such as machining, extrusion, etc. This has led to an entire set of tools separate from the basic CAD operations used to design detail parts and generate manufacturing drawings. As an illustration of the magnitude of this issue, between the McDonnell Douglas, Northrop, and Dassault lofting tools, there at least 71 different surface types. This number is continuing to increase as new surface types are introduced to reflect new design and manufacturing proceses.

This makes porting CAD models between systems very difficult. It is prohibitively time consuming to restrict the designer to surface types that are common to multiple systems. These unique surface types were added to improve the design process and it would be a step backwards not to use them. A second approach is to generate conversions of surfaces of one type to another. The most common choice is NURBS since many surfaces, such as parametric bicubics, conic surfaces, and surfaces of revolution can be represented exactly. However, many of these unique surface types are procedurally rather than mathematically defined. An example of a procedural surface is the Spine Controlled Dependent Fillet Surface with Functional Radius. In this surface, circular elements lie in planes normal to a controlling space curve, called the spine curve. The surface defines a fillet between any two supporting surfaces with the radius of any given element defined by a function. The surface is a dependent procedural surface that conforms exactly, in both position and tangency conditions with the supporting surfaces. The radius function can be defined by any of three methods, the most common being interpreted as a function of percent of arc length of the spine curve over the defined range of the fillet surface.

## PARAMETRIC EVALUATOR APPROACH

An alternate approach is to use a parametric evaluator[12,13] for interogating analytic surfaces. The basis for the parametric evaluator approach is defining a set of two-dimensional curvilinear coordinates to represent each surface. In defining this coordinate system, each coordinate direction can be restricted to the range from zero to one, such that any point on the surface has an associated unique coordinate pair in this range. The parametric evaluator is the internal module of the CAD system which takes a parametric coordinate and returns the physical coordinates for the associated point on the surface. Parametric evaluators have been written as stand-alone modules for several CAD and lofting systems including Dassault CATIA, EDS Unigraphics II, McDonnell Douglas CADD/CALL and GOLD, and Northrop-Grumman NCAD/NCAL. Having parametric evaluators for each surface type, all that remains is to write tools that access these surfaces to operate on the unit parametric square. This is accomplished through the Advanced Integrated Mathematical System (AIMS) software. This software, developed by McDonnell Douglas, includes a variety of capabilities for surface interrogation, intersection, etc. All of the dependencies based on the surface type are transparent to the user. Internally, AIMS determines the surface type and accesses the appropriate parametric evaluator.

AIMS does make use of surface type information in specific cases. Operations on certain types of surfaces may produce an exact analytic solution. For example, in intersecting a plane with conic section there is an exact solution which will be a conic curve. AIMS will use this information when appropriate to improve speed and/or accuracy transparent to the user.

To allow Unigraphics to access data from other CAD systems, a major effort was undertaken in 1991 by a group at McDonnell Douglas to implement AIMS into Unigraphics. As of Version 9, in addition to its internal entity types, Unigraphics can access surfaces from CATIA, CADD/CALL, GOLD, and NCAD/NCAL through AIMS and the appropriate parametric evaluator. Through AIMS, Unigraphics can perform operations, such as the intersection shown in Figure 1, on surfaces from other CAD systems without converting the surface to another type. Support for surfaces from another CAD system would only require the development and linking in of a stand-alone parametric evaluator for its surface types.

## IMPLEMENTATION IN MACGS

The McDonnell Aircraft Computational Grid System (MACGS)[14,15,16] is a general purpose interactive grid generation system which has capabilities for geometry manipulation, and generation of structured, overlapping, and unstructured grids. Implementation of AIMS into MACGS eliminates the need to obtain a point definition of a geometry from the CAD system. This is significant since acquiring a point definition geometry for a full aircraft configuration can take up to two weeks.

Several options were considered in implementing analytic surfaces in MACGS. MACGS stores the user created surfaces and grids in a working file. Storing analytic surface definitions in this file as well was considered but this could require a lot of file space for a large CAD model. The method chosen was to have two files open. In addition to the users working file, the user can also open a Unigraphics part file. There are two ways that the analytic surfaces of the CAD model can be stored in the Unigraphics part file. Surfaces which are created using Unigraphics or read into Unigraphics from other CAD systems through IGES files or some other format are stored in the Unigraphics file directly. The true LOFT surfaces are stored in a separate database

and the Unigraphics part file will contain only a reference to these surfaces. Unigraphics user functions are used to interrogate the part file to get a list of surfaces. Through AIMS, access to these surfaces is transparent to the user whether they are stored in the part file, or in the LOFT database which may reside on a different physical system and is accessed via the network. The dashed surfaces in Figure 2, which were defined in the older McDonnell Douglas CAD system, CADD, are accessed from the loft data base. The solid line surfaces reside in the Unigraphics part file along with references to the loft surfaces. Since Unigraphics user functions are used to access the Unigraphics part file, Unigraphics must be running on the grid generation platform.

The AIMS software is used to retrieve points along the edges of the surfaces which are used to display the surfaces. The aircraft forebody configuration of Figure 2 is accessed from the Unigraphics part file and displayed in MACGS as shown in Figure 3. The number of points displayed on an edge is determined by a maximum chordal tolerance between the actual surface and the straight segment between successive points. The user can generate structured or unstructured grids on an analytical surface by specifying distributions of points along each edge of the surface. Parametric coordinates along the edges of the surface are computed from these distributions and a grid is generated in parametric coordinates. Then AIMS is used to compute the corresponding physical coordinates of the grid points. Structured and unstructured grids generated on an analytic surface are shown in Figures 4 and 5 respectively.

Often, there is a need to manipulate geometry[17]. Doing this in the CAD system involves getting a CAD designer involved or extensive CAD training for the CFD engineer. Building capabilities to manipulate the analytic surfaces within the grid system would add another level of complexity and increase the required user skill level dramatically. A simpler method is to use tools that already exist in the MACGS grid generation system. MACGS capabilities include splitting, combining, smoothing, etc., for surfaces in an interactive graphical environment. Since the user is modifying the geometry for a discrete analysis, some degree of approximation is acceptable.

The capability to project user-created geometry or grids onto one or multiple analytic surfaces was also incorporated using AIMS functionality. A point definition surface representation is used to define grids and surfaces generated within MACGS and interpolation on a curve fit in each direction is performed to redistribute points on the surface. This requires care by the user since inappropriate operations can introduce variation from the original surface. To ensure that points are precisely on the CAD geometry, the new grids can be projected back onto the original CAD surfaces at any point. Figure 6 illustrates projection of a structured grid onto multiple analytic surfaces.

## COMPENSATING FOR ANALYTIC SURFACE PARAMETERIZATION

It is desirable to have grid distributions based on the physical arclength of a surface, rather than its parametric coordinates. This is especially true for analytic surfaces. There may be a large range of parameterizations for a given surface and the parametric coordinates are generally non-linear with respect to arclength on the surface. Therefore some method is needed to assure that the distribution in physical space satisfies the desired spacing, stretching, and smoothness characteristics.

This same problem has been addressed for surfaces defined by a set of discrete points. When interpolating between the discrete points, some type of fit is assumed, such as a bi-cubic spline. The distribution of the discrete points can be thought of as the surface parameterization. For

example, redistributing points on a surface defined by points clustered toward one edge or with a pronounced variation in the distribution across the surface can yeild an undesirable result if some method to base the distribution of points on arc length along the surface is not used.

This is accomplished for analytic surfaces by first calculating a distribution of points on the surface, uniform in parametric space. The arclength distribution of these points is used to define a mapping from the desired arclength parametric coordinates to coordinates based on the parameterization of the surface. Since all physical points are generated by evaluating parametric coordinates on the CAD surface description using AIMS, all points are guaranteed to lie on the surface. The intermediate mapping affects only where they lie on the surface. Also, points on the bounds of the parametric space will lie on the edge of the surface.

Generation of a uniform distribution of points is illustrated in Figure 7. Figure 7b shows the effect of applying a uniform distribution in the parametric coordinates of the analytic surface. Note that the spacing along the body is larger near the singularity than at the opposite edge. Figure 7c represents the desired uniform distribution in an arc length space. To achieve this uniform distribution, the point distribution in the analytic surface parametric coordinates shown in Figure 7d is evaluated to produce the more uniform grid of Figure 7e. This effect of the surface parameterization can be even more pronounced for unstructured grids as shown in Figure 8. Figure 8b shows the problem that the parameterization near the singularity can have on the unstructured grid generation algorithm when compensation is not used. To overcome this problem for unstructured grids, it is useful to generate the grid in a parametric space that more closely approximates the shape of the surface, as shoen in Figure 8c. Figures 8d and 8e again show the intermediate mappings that produce the surface grid presented in Figure 8f.

## RESULTS

The initial implementation provides a useful capability to read surfaces and put a distribution of points on the surfaces. There are still several issues which are being addressed. One of the most important deals with trimmed surfaces. Currently, for a trimmed surface the entire untrimmed surface is read. Trimmed surfaces can be very general in nature with multiple holes and irregular shaped cut-outs. It is often not possible to come up with a single unit parameterization of only the trimmed part of the surface. Therefore it is necessary to maintain the analytic definitions of the surface trim curves so that the grid can conform to thes curves as well as the shape of the surface. Future enhancements will also bring in the trimming information. Another important related function is the ability to deal with analytic curves extracted from the edge of a surface. These curves are necessary for constructing the edges of a single grid which spans multiple surfaces.

Manipulations inside the grid generation system on a point-definition surface created from the analytic surface, are based on a fit through the points. This is not generally a problem since the fit through the surface points is smooth and deviation from the surface is usually small compared to the spacing of points on the surface. When necessary, these surfaces can be projected back to the analytic surface.

Another issue involves speed and memory requirements. Memory for the analytic surfaces is allocated dynamically as the surfaces are accessed from the file. When working with a lot of surfaces, there is a tradeoff between keeping everything in memory (and often hitting a memory limit), and cleaning up memory after each operation (and paying a penalty for increased time to

access the surface (sometimes across a network) each time it is needed.

For a model with a large number of surfaces, it is desirable to generate a grid spanning multiple surfaces, rather than gridding each one and combining them. This can be accomplished by generating a grid from edge curves and then projecting this grid onto the analytic surfaces. However, the distribution of the grid points after projecting may not match the distribution of the original surface, particularly when the surfaces have a lot of curvature. This is especially important for the boundary spacing. This is generally not as critical for newer models developed early in the design phase, since these often have fewer surfaces defining the geometry. But we have observed that as the maturity of the design increases, the complexity of the surface definition and the number of surfaces defining it increase rapidly. As a result, developing the capability to generate grids across multiple surfaces directly is high on our priorities.

There were several benefits to implementing support for analytic surfaces using AIMS. First of all, the package is very modular and was incorporated rather easily. Since memory for the analytic surfaces is allocated dynamically within AIMS, no major changes to the MACGS data structures were required. The biggest issues in implementing AIMS were related to the linker. The interrelation between AIMS and Unigraphics is at times confusing because some AIMS routines call UG routines and other UG routines call AIMS routines, and it took some effort to sort this out. Secondly, AIMS is robust and users know they have the exact surfaces from the CAD model. Also, the accuracy of operations such as projection are consistent with the equivalent operations performed within the CAD environment.

## CONCLUSIONS

Implementation of the AIMS routines into MACGS provides the capability to access any surface in a Unigraphics model, regardless of type. This greatly simplifies the process of geometry acquisition for CFD analyses. However, the amount of manipulation of the geometry within the grid generation system depends on several things including the quality of the model (gaps, overlap, etc.) and simplifications to be made to the geometry for analysis reasons. This approach worked well due to the availability of a mature library for accessing surfaces and evaluating them parametrically. This proved to be much simpler than converting a vast range of surfaces to a particular type, such as NURBS, and making extensive internal changes to the grid generation system to support this type.

## REFERENCES


1. Lijewski, L.E., Cipolla, J., Thompson, J.F., and Gatlin, B., "Program EAGLE User's Manual," AFATL-TR-88-117, Vols. I-III, September 1988.

2. Steinbrenner, J.P., Chawner, J.R., and Fouts, C.L., "The GRIDGEN 3D Multiple Block Grid Generation System," WRDC-TR-90-3022, Vol. I,II, July 1990.

3. Remotigue, M.G., Hart, E.T., and Stokes, M.L., "EAGLEView: A Surface and Grid Generation Program and its Data Management," NASA CP 3143, pp. 243-251, April 1992.

4. Lohner, R., and Parikh, P., "Generation of Three-Dimensional Unstructured Grids by the Advancing-Front Method." AIAA Paper No. 88-0515, January 1988.

5. Parikh, P., Pirzadeh, S., and Frink, N.T., "Unstructured Grid Solutions to a Wing/Pylon/Store Configuration Using VGRID3D/USM3D," AIAA Paper No. 92-4572, August 1992.

6. Weatherill, N.P., "A Method for Generating Irregular Computational Grids in Multiply Connected Planar Domains," International Journal for Numerical Methods in Fluids, Vol. 8, pp. 181-197, 1988.

7. Benek, J.A., Steger, J.L., and Dougherty, F.C., "A Flexible Grid Embedded Technique with Application to the Euler Equations," AIAA Paper No. 83-1944, July 1983.

8. Baysel, O., Fouladi, K, and Lessard, V.R., "Multi-grid Upwind Viscous Flow Solver on Three-Dimensional Overlapped and Embedded Grids," AIAA Journal, Vol. 29, No. 6, 1991.

9. Smith, B., and Wellington, Jr., "Initial Graphics Exchange Specification (IGES), Version 3.0," NBSIR 86-3359, April 1986.

10. "U.S. Product Data Association and IGES/PDES Organization Reference Manual," October 1994.

11. Piegl, L., "On NURBS: A Survey," IEEE Computer Graphics and Applications, Vol. II, No. 1, 1991.

12. Melson, T.G., "Analytic/Parametric Conversions," Proceedings CAM-I Seminar on Geometric Representation - Which Way to Go?, Tucson, Arizona, P-84-MM-01, pp. 184-206, January 1984.

13. Moreau, P., "Applying Object-Oriented Architecture to Industrial Geometric Modelers," International Symposium on Advanced Geometric Modelling for Engineering Applications, Berlin (West), FRG, pp. 365-374, November 1989.

14. Gatzke, T.D., LaBozzetta, W.F., Finfrock, G.P., Johnson, J.A., and Romer, W.W., "MACGS: A Zonal Grid Generation System for Complex Aero-Propulsion Configurations," AIAA Paper No. 91-2156, June 1991.

15. Gatzke, T.D., and Weatherill, N.P., "Unstructured Grid Generation Using Interactive Three-Dimensionsl Boundary and Efficient Three-Dimensional Volume Methods," AIAA Paper No. 93-3452, August 1993.

16. LaBozzetta, W.F., Gatzke, T.G., Ellison, S., Finfrock, G.P., and Fisher, M.S., "MACGS - Toward The Complete Grid Generation System," AIAA Paper No. 94-1923, June 1994.

17. Gatzke, T.D., LaBozzetta, W.F., Cooley, J., and Finfrock, G.P., "Geometry Acquisition and Grid Generation -- Recent Experiences with Complex Aircraft Configurations," NASA CP 3143, pp. 31-43, April 1992.

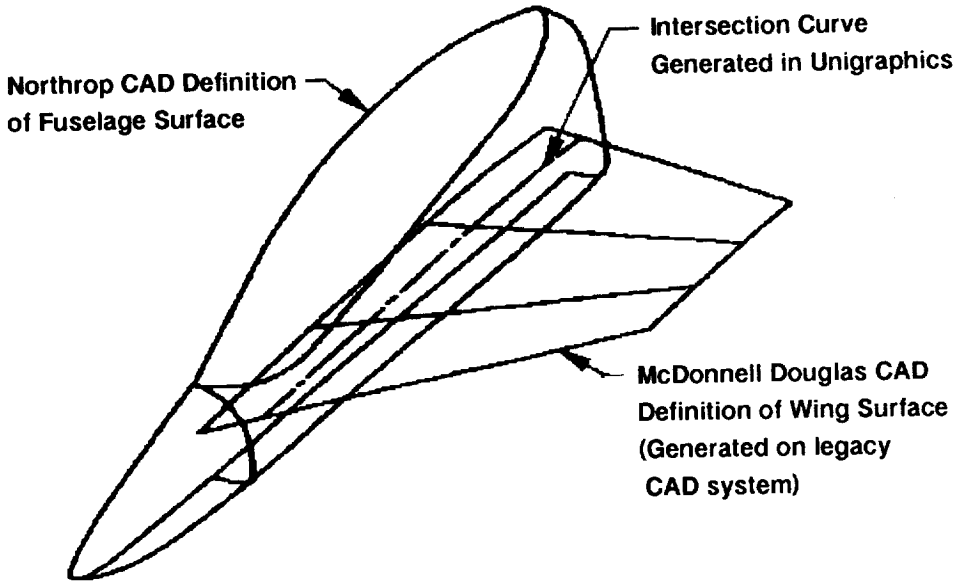



**Figure 1. – Unigraphics Intersection of Surfaces from Different CAD Systems Using Parametric Evaluators.**

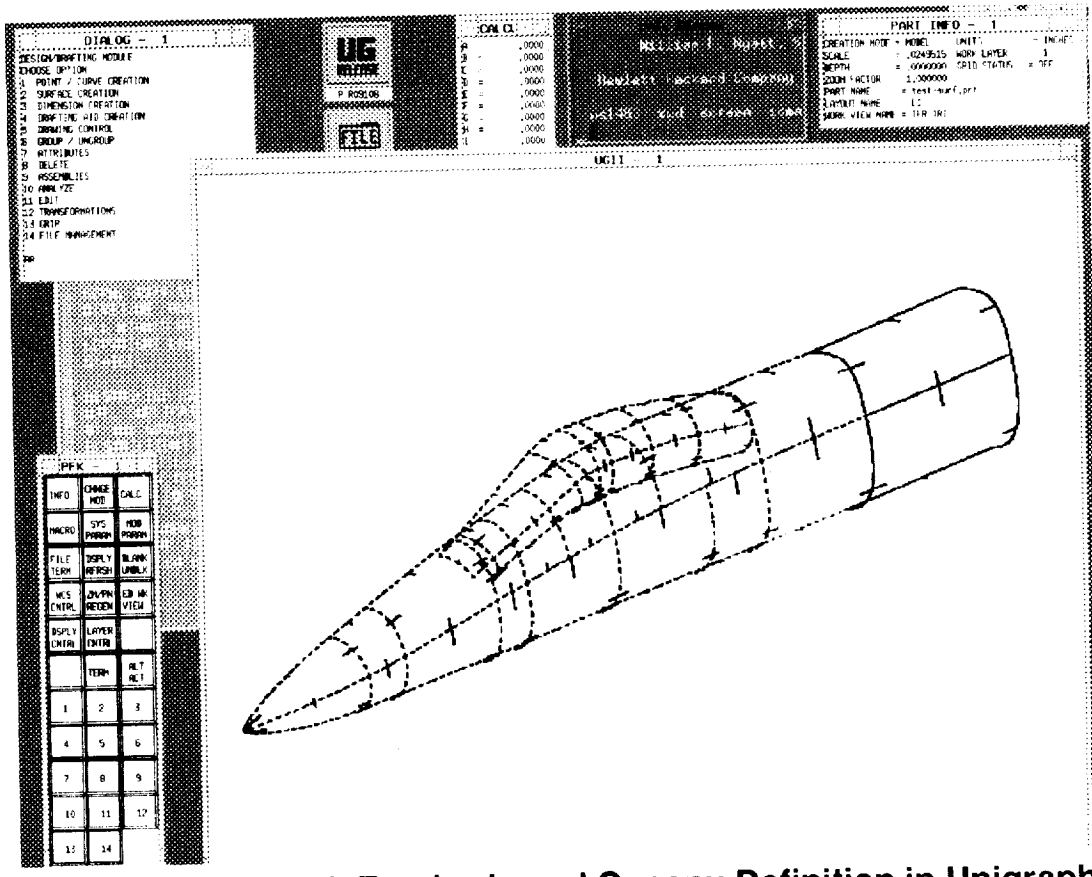


**Figure 2. – An Aircraft Forebody and Canopy Definition in Unigraphics.**

**(Dashed surfaces defined in legacy CAD system.)**

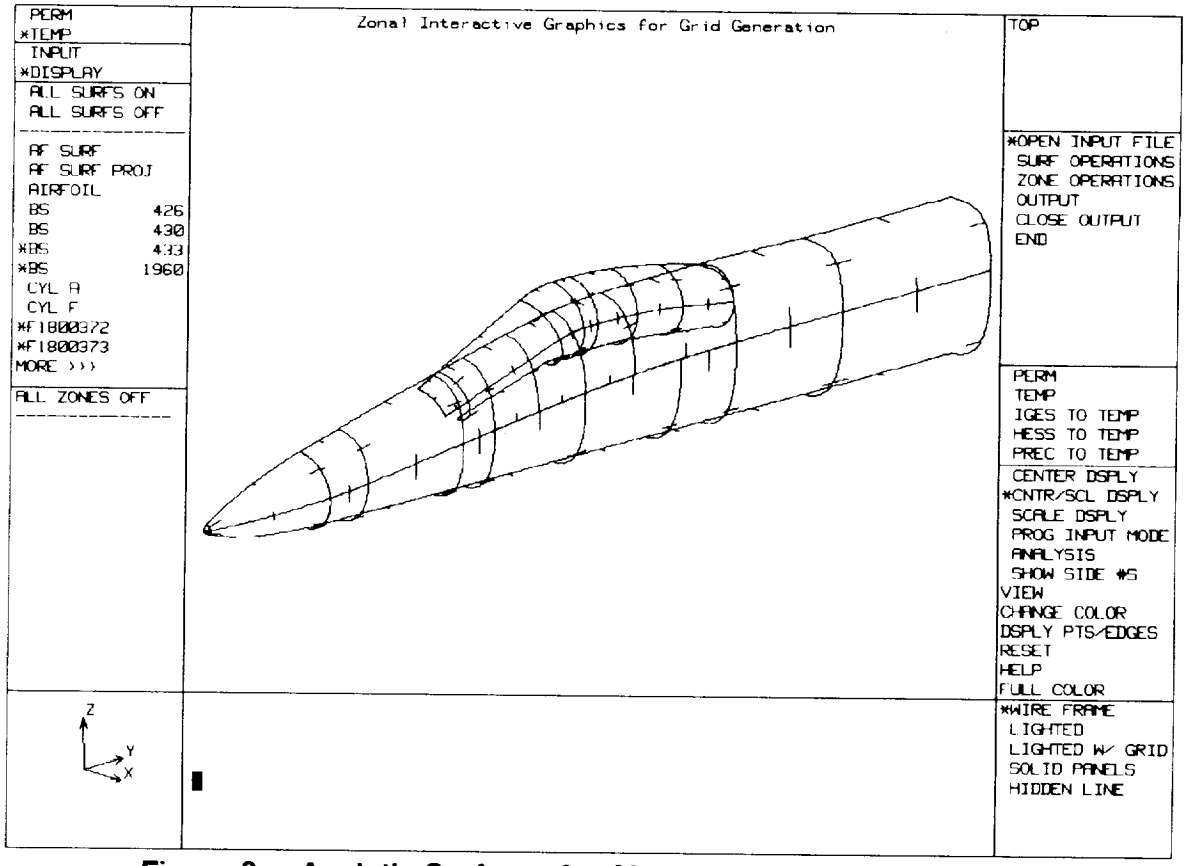



**Figure 3. – Analytic Surfaces for Aircraft Forebody and Canopy Displayed in MACGS Surface Gridding Module, ZONI3G.**

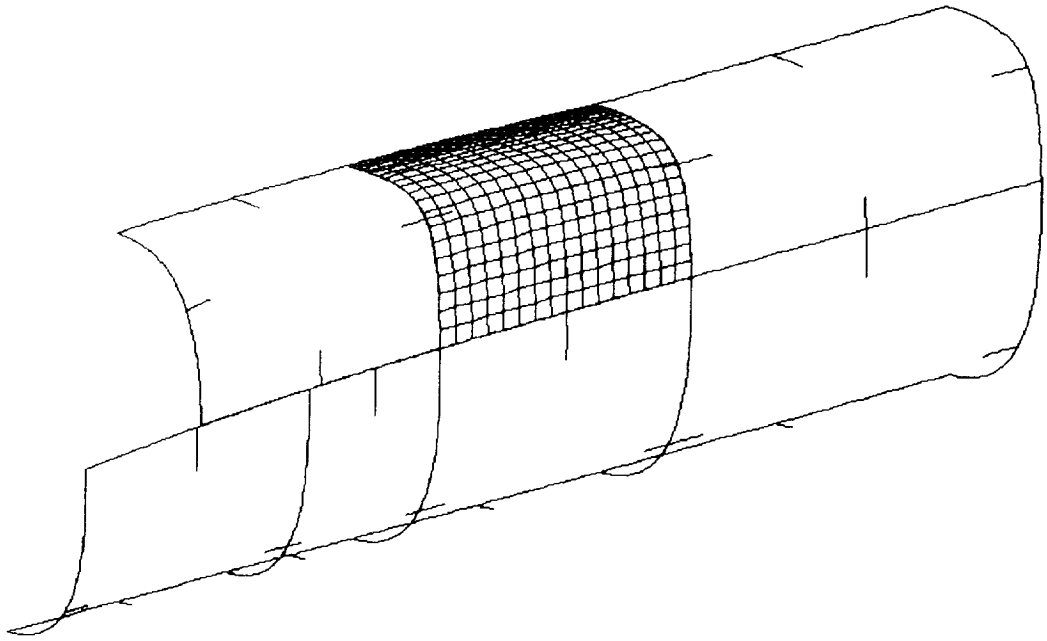


**Figure 4. – Structured Grid Generated Directly on Analytic Surface with Clustering Toward Top of Fuselage.**

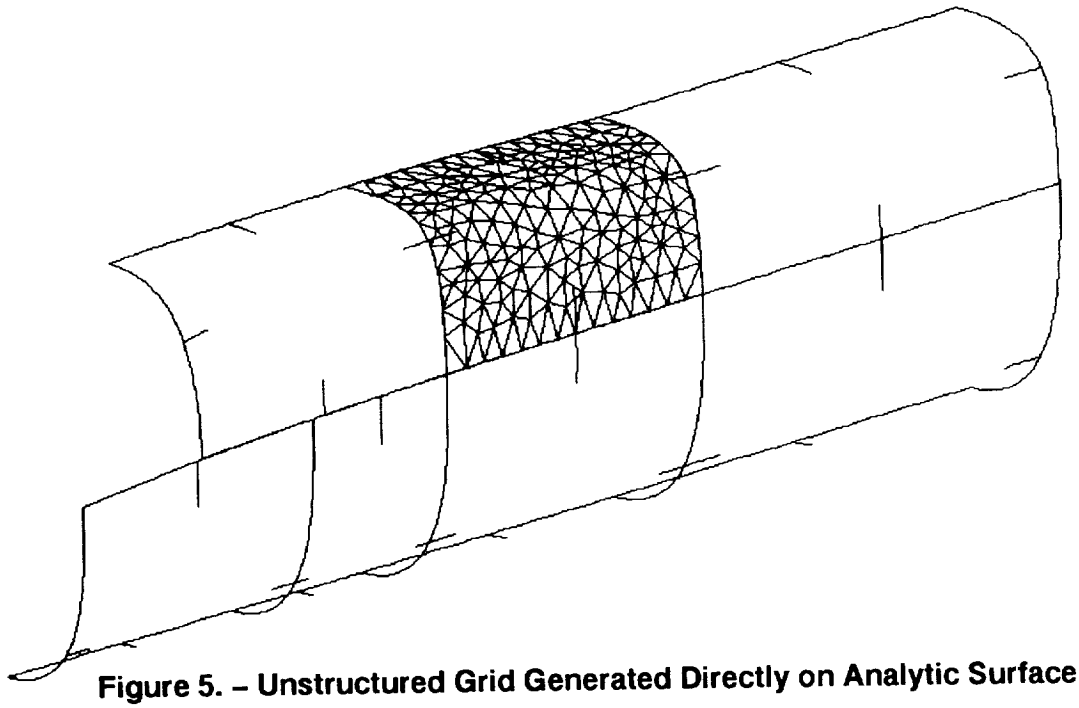


Figure 5. – Unstructured Grid Generated Directly on Analytic Surface.

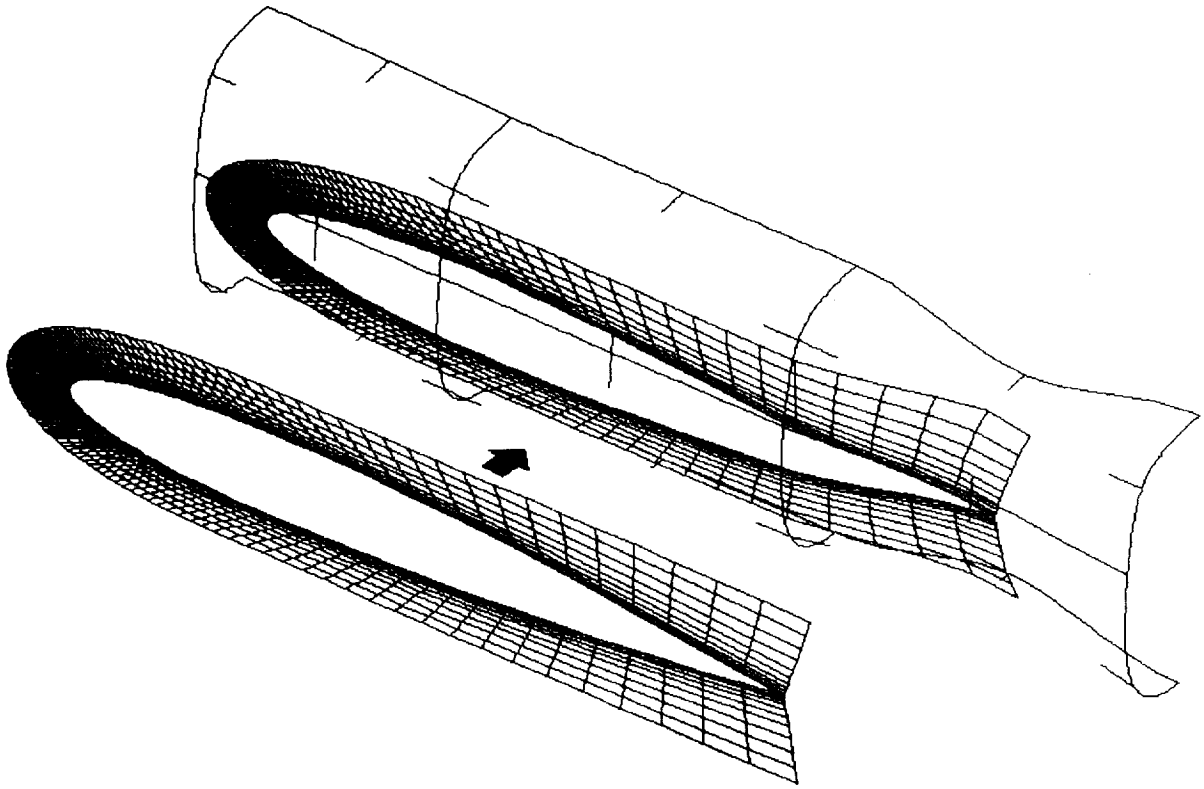


Figure 6. – Structured Grid Projected onto Multiple Analytic Surfaces.

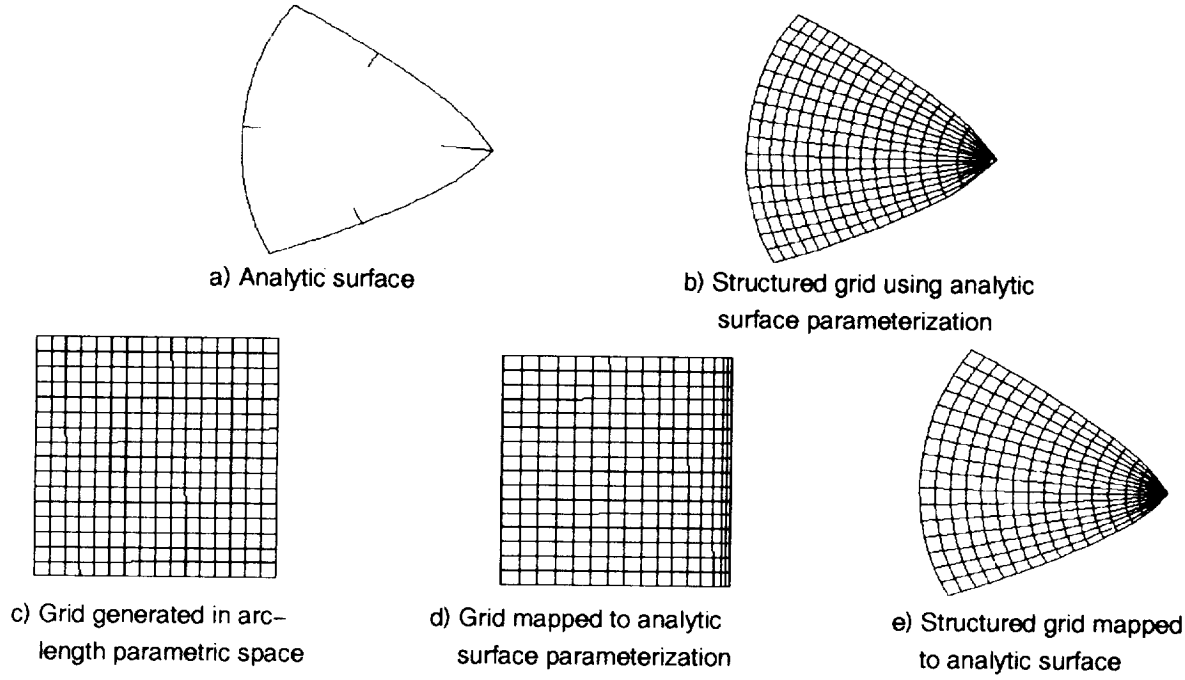



**Figure 7. – Structured Grid Generation: Compensation for Analytic Surface Parameterization.**

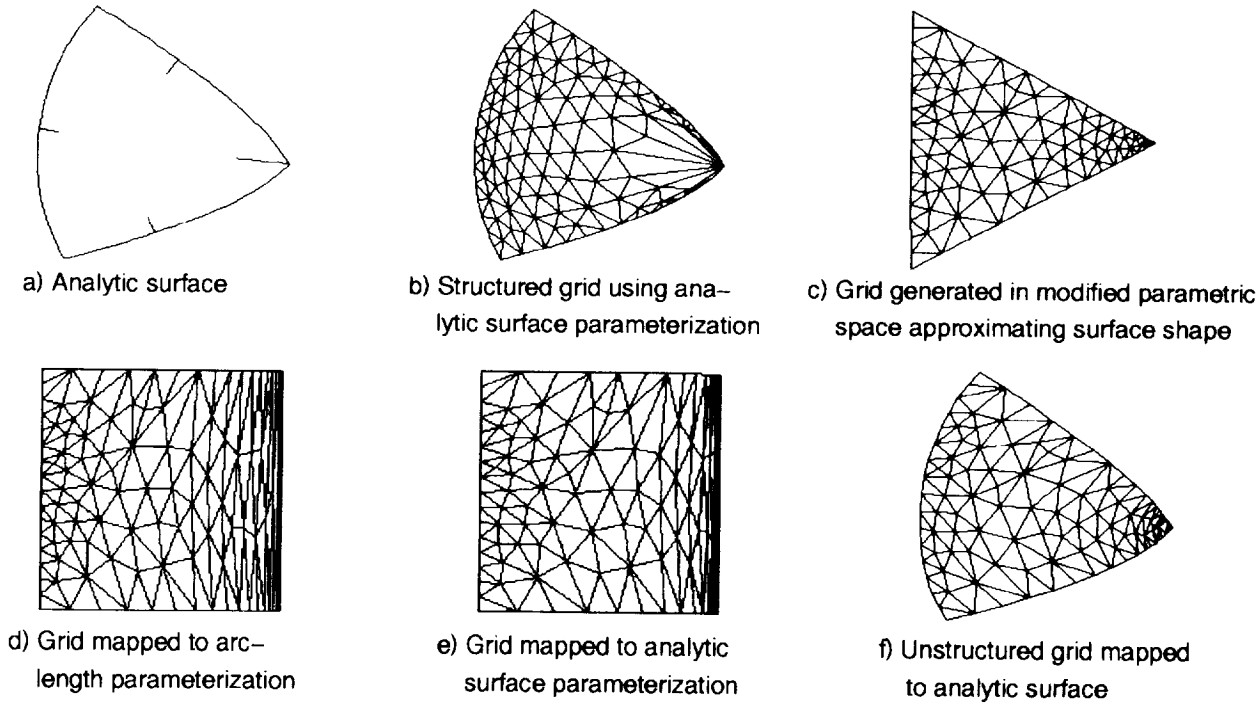



**Figure 8. – Unstructured Grid Generation: Compensation for Analytic Surface Parameterization.**

N95-28755

# GRID GENERATION ON TRIMMED BEZIER AND NURBS QUILTED SURFACES

Chung-Jin Woan, Willard C. Clever, and Clement K. Tam
North American Aircraft Division/Rockwell International
Seal Beach, California 90740-7644

## SUMMARY

This paper presents some recently added capabilities to RAGGS, Rockwell Automated Grid Generation System. Included are the trimmed surface handling and display capability and structured and unstructured grid generation on trimmed Bezier and NURBS quilted surfaces. Samples are given to demonstrate the new capabilities.

## INTRODUCTION

As described in Ref. 1, RAGGS (Rockwell Automated Grid Generation System) provides an integrated CFD (Computational Fluid Dynamics) environment where a user can 1) import a CAD (Computer Aided Design) generated configuration or a geometry from other sources, 2) build a CFD grid directly on the imported geometry, 3) set up flow solver input, 4) postprocess the flow solution, and 5) display the flow solution data and geometry simultaneously. The program structure of RAGGS is depicted in Fig. 1. It consists of a series of modules to handle various tasks in the CFD process: starting with a given geometry, usually input as an IGES (Initial Graphics Exchange Specification) file, through construction of CFD geometry database, domain blocking, surface grid generation, volume grid generation, flow simulation, to postprocessing of the flow solution to obtain data useful for engineering design and analysis. A file containing the geometry data read by RAGGS is referred to as a RASCAL (Rockwell Automated System for Computer Aided Lofting) file. As shown in Fig. 1, each module is connected to the common database shown in the central hub. Through the GUI (Graphical User Interface) module and/or grid file module, RAGGS enables user to quickly communicate with some state-of-the-art flow solvers, flow analysis software, and grid generators as listed in the figure. The RAGGS surface grid generation is operated directly on the quilts of rational Bezier patches and/or NURBS (Non Uniform Rational B-Spline), each of arbitrary order. This minimizes the interaction of CAD/CFD groups and eliminates the need of re-constitution of the imported geometry, which often requires hundreds of hours of manipulation to achieve a data format suitable for the grid generation software package.

This paper presents some recently added capabilities to RAGGS. These include trimmed surface/curve handling and display, unstructured surface grid generation, and structured/unstructured grid generation on trimmed Bezier and NURBS quilted surfaces. The materials presented in this paper are intended to complement that given in Ref. 1.

## GEOMETRY DATABASE

The coordinates of a surface in space can be described in terms of two parametric variables. A patch is a mapping from a space defined by these two parametric variables to the three Cartesian coordinates in 3-D space. The untrimmed domain of each patch is the rectangular region in parameter space, $s_0 \leq s \leq s_1$ and $t_0 \leq t \leq t_1$, where usually $s_0 = t_0 = 0$ and $s_1 = t_1 = 1$. Additionally, by defining active and inactive regions, trim curves may be used to further restrict the domain of a patch. These trim curves are nonintersecting closed loops of Bezier curves or NURBS in the parametric space of a patch.

For a general configuration, aircraft or otherwise, there is no uniform parameterization which can describe the entire surface. However, the surface can be described by a set of individually parameterized patches,



PRECEDING PAGE BLANK NOT FILMED

possibly with trim curves. Any subset of these patches, having surface continuity, can be said to constitute a quilt. Any configuration or any number of configurations can be completely described by a number of such quilts. RAGGS uses such quilts of rational Bezier patches and/or NURBS, each of arbitrary order, to create grids which lie exactly on the parameterized surface. These patches may have 3 or 4 sides and need not be distributed in any type of uniform manner.

## Rational Bezier Patch and NURBS

Bezier and NURBS curves in 3-D space are composed of a point which varies parametrically with a single parameter. Surfaces are composed of a curve which itself varies parametrically in 3-D space according to another parameter. Consider a such NURBS curve, of degree $n$, parameterized in t. The curve may be written in terms of the B-Spline basis functions as follows:

$$\vec{G}(t) = \frac{\sum_{j=0}^{K_n} v_j \vec{g}_j b_j^n(t)}{\sum_{j=0}^{K_n} v_j b_j^n(t)} \qquad \begin{cases} v_j & - \text{ the weight coefficients} \\ \vec{g}_j & - \text{ the control points} \\ b_j^n(t) & - \text{ the B-Spline basis functions} \end{cases} \tag{1}$$

A knot vector is defined by the nondecreasing sequence $t_{-n}, \ldots, t_0, t_1, \ldots, t_{K_n}, t_{K_n+1}$. The B-Spline basis functions are defined by the following recursion,

$$b_i^{n+1}(t) = \frac{t - t_{i-n-1}}{t_i - t_{i-n-1}} b_{i-1}^n(t) + \frac{t_{i+1} - t}{t_{i+1} - t_{i-n}} b_i^n(t) \qquad \text{where,} \quad \mathrm{b_i^0(t)} = \begin{cases} 1 & t_i \le t \le t_{i+1} \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

The basis functions have the property that

$$\sum_{j=0}^{K_n} b_j^n(t) \equiv 1 \tag{3}$$

For a rational Bezier curve, $K_n = n$, is equal to the degree of the curve, the parameter domain becomes $[0, 1]$, the knot vector becomes $t_{-n}, t_{-n+1}, \ldots, t_0 = 0$, and $t_1, \ldots, t_n, t_{n+1} = 1$, and the basis functions become the Bernstein functions,

$$b_k^n(t) = \frac{n!}{k!(n-k)!}(1-t)^{n-k} t^k \tag{4}$$

If we let the weights, $v_j$, and the control points, $\vec{g}_j$, be interpolated functions of another parameter $s$,

$$v_j(s) = \sum_{i=0}^{K_m} w_{ji} b_i^m(s) \qquad \vec{g}_j(s) = \frac{\sum_{i=0}^{K_m} w_{ji} \vec{p}_{ji} b_i^m(s)}{\sum_{i=0}^{K_m} w_{ji} b_i^m(s)} \tag{5}$$

Then Eq. (1) together with Eq. (5) constitutes a surface patch parameterized in $s$ and $t$ as follows:

$$\vec{P}(s,t) = \frac{\sum_{i=0}^{K_n} v_j(s) \vec{g}_i(s) b_i^n(t)}{\sum_{i=0}^{K_n} v_j(s) b_i^n(t)} = \frac{\sum_{j=0}^{K_n} \left[ \sum_{i=0}^{K_m} w_{ji} b_i^m(s) \frac{\sum_{i=0}^{K_m} w_{ji} \vec{p}_{ji} b_i^m(s)}{\sum_{i=0}^{K_m} w_{ji} b_i^m(s)} \right] b_j^n(t)}{\sum_{j=0}^{K_n} \left[ \sum_{i=0}^{K_m} w_{ji} b_i^m(s) \right] b_j^n(t)} = \frac{\sum_{j=0}^{K_n} \sum_{i=0}^{K_m} w_{ji} \vec{p}_{ji} b_i^m(s) b_j^n(t)}{\sum_{j=0}^{K_n} \sum_{i=0}^{K_m} w_{ji} b_i^m(s) b_j^n(t)} \tag{6}$$

with $\vec{p}_{ij}$ being the control points and $w_{ij}$ the weights.

For fixed $t$, Eq. (6) may be written as

$$\vec{H}(s) = \vec{P}(s,t) = \frac{\sum_{i=0}^{K_m} \left[ \sum_{j=0}^{K_n} w_{ji} b_j^n(t) \frac{\sum_{j=0}^{K_n} w_{ji} \vec{p}_{ji} b_j^n(t)}{\sum_{j=0}^{K_n} w_{ji} b_j^n(t)} \right] b_i^m(s)}{\sum_{i=0}^{K_m} \left[ \sum_{j=0}^{K_n} w_{ji} b_j^n(t) \right] b_i^m(s)} = \frac{\sum_{i=0}^{K_m} u_i(t) \vec{h}_i(t) b_i^m(s)}{\sum_{i=0}^{K_m} u_i(t) b_i^m(s)} \tag{7}$$

with

$$u_i(t) = \sum_{j=0}^{K_n} w_{ji} b_j^n(t) \qquad \vec{h}_i(t) = \frac{\sum_{j=0}^{K_n} w_{ji} \vec{p}_{ji} b_j^n(t)}{\sum_{j=0}^{K_n} w_{ji} b_j^n(t)} \tag{8}$$

Equations (7)-(8) define another set of curves parameterized in s for fixed t, i.e., for fixed $t$,

$$\vec{H}(s) = \frac{\sum_{i=0}^{K_m} u_i(t) \vec{h}_i(t) b_i^m(s)}{\sum_{i=0}^{K_m} u_i(t) b_i^m(s)} \tag{9}$$

with $u_i$ being the weights, $\vec{h}_i(t)$ the control points, and $\vec{b}_i^m(s)$ the B-spline basis functions.

In RAGGS, all the surface grid generators are formulated in the 3-D physical space, i.e., the grid points are computed directly in terms of the three Cartesian coordinates, $x$, $y$, and $z$. These space points are then placed exactly on the nearest quilted surface point by iterating for the patch number and parametric variables. A key element in a formulation using discrete patches is the boundary data, which is a description of how the parametric variables change as one crosses from the boundary of one patch to the boundary of an adjacent patch.

## Definition Of Active Region Of Patch

As mentioned before, a surface is descibed by a quilt of Bezier patches or NURBS, possibly with trim curves. Each patch is parameterized in $s$ and $t$. The untrimmed domain of each patch is $s_0 \leq s \leq s_1$ and $t_0 \leq t \leq t_1$, where usually $s_0 = t_0 = 0$ and $s_1 = t_1 = 1$. Trim curves (if any) are nonintersecting Bezier curves or NURBS forming closed loops in the parametric space of a patch.

$$\begin{Bmatrix} s \\ t \end{Bmatrix} = \vec{C}_i(\sigma) \qquad i = 1, \ldots, n \tag{10}$$

Here, $\vec{C}_i(\sigma)$ is the $i$th parameterized trim curve, $\sigma$ is the parameter, and n is the number of trim curves.

The entire domain of an untrimmed patch is defined to be active. Patches which have trim curves have both active and inactive regions. By convention, the active region of a trimmed patch is the region that lies

on the left of each trim curve as a point advances along the curve. Here, left of a trim curve at a point $\vec{C}_i(\sigma)$ is the direction of the vector formed as the cross product (in the order specified) of the surface normal and the tangent vector to the trim curve at $\vec{C}_i(\sigma)$.

## Patch Boundary Data

The patch boundary data is a description of how the parametric variables change as one crosses from the boundary of one patch to the boundary of an adjacent patch. Boundary data for each patch is given in terms of a series of parameter ranges for each side. Figure 2 shows a typical boundary connectivity between two adjacent patches (in this case, patches $p$ and $q$). For patch $p$, there may be data for ranges,

$$\begin{aligned} s_{b_1} \leq\ & s \leq s_{b_2} \\ s_{b_3} \leq\ & s \leq s_{b_4} \\ & \vdots \end{aligned} \tag{11}$$

For each parameter range, an adjacent patch number and a connectivity matrix of six numbers are given. In Fig. 2, we assume that a point has crossed the boundary at $(s_b, t_b)$ with

$$t_b = t_1, \quad s_0 \leq s_b \leq s_1 \tag{12}$$

Then the boundary data for the side $t_b = t_1$ and the range which includes $s_b$ will give the new patch number $q$ and the corresponding parameter values of the boundary point $(t_b, s_b)$ on this patch $q$ as follows:

$$\begin{pmatrix} s_b \\ t_b \end{pmatrix}_{patch\ q} = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \end{bmatrix} \begin{pmatrix} s_b \\ t_b \\ 1 \end{pmatrix}_{patch\ p} \tag{13}$$

where $a_{ij}$ are the transformation matrix coefficients.

## Trim Curve Boundary Data

Consider the trim curves of Eq. (10). Analogous to the patch boundaries, boundary data for trim curves is given for a series of parameter ranges of the curve being crossed as follows:

$$\begin{aligned} \sigma_{b_1} \leq\ & \sigma \leq \sigma_{b_2} \\ \sigma_{b_3} \leq\ & \sigma \leq \sigma_{b_4} \\ & \vdots \end{aligned} \tag{14}$$

When crossing trim curve boundaries, the boundary data which includes the parameter range of the crossing point $\sigma_b$ will give the new patch and the corresponding trim curve on the new patch, and the parameter value of the boundary point on the corresponding trim curve as folows:

$$(\sigma_b)_{trim\ curve\ on\ patch\ q} = \begin{bmatrix} a_1 & a_2 \end{bmatrix} \begin{pmatrix} 1 - \sigma_b \\ \sigma_b \end{pmatrix}_{trim\ curve\ on\ patch\ p} \tag{15}$$

where $a_1$ and $a_2$ are the transformation matrix coefficients.

## Geometry Creation and IGES/RAGGS Interface

The basic method of geometry input is through IGES files created by any CAD system. The data in the IGES file must be translated into surfaces in the form of NURBS and Bezier Patches. These are the fundamental surface forms used by RAGGS. Translating the data into the RAGGS internal format may take a number of different paths, depending on the type of data in the IGES file. A flow diagram of the process is shown in Fig. 3. The processing of the IGES file yields four different types of information which are stored for later use.

NURBS, entity type 128, contain the only type of surface data which may be used directly. The control points and other data for these entities can be read directly from the IGES file. For other surface entity types such as planes, parametric splines, surfaces of revolution, tabulated cylinders, and ruled surfaces, the data must be converted into a NURBS or rational Bezier patch format. In all cases the parameter conversion results in a surface which is mathematically equivalent to the original surface.

The second type of information used by RAGGS is information about curves. Curve entities may be in the form of parametric splines, NURBS, or point data. The curve data may be used subsequently for trim curves, surface generation, (surfaces of revolution, tabulated cylinders, ruled surfaces, etc.) or they may have other uses. Surface generation using point data involves curve fitting. This may result in a surface which is not exactly equivalent to the surface used to generate the points.

Trimmed surface data is the third type of information which is retained and processed for later use. When combined with the surface and curve data, this data may be used to create the information necessary to define the active and inactive domains on each patch. This allows the trimmed surfaces to be displayed and to create grids which cross the trim boundaries.

Ruled surface information is the fourth type of data which is stored. This data is combined with the curve data to create additional surfaces parameterized in the form of Bezier patches or NURBS.

A file containing the patch information and the related boundary data read by RAGGS is referred to as a RASCAL file.

# NUMERICAL TOOLS

Various numerical tools which are fundamental to our grid generations are described herein.

## Projecting A Point On A (Trimmed) Surface

A point on a quilted surface may be identified by $(p, s, t)$ with its position vector computed by $\vec{R}(p, s, t)$. $p$ is the patch number. For a given patch number $p$, $s$ and $t$ are the patch parametric variables and $\vec{R}$ is a Bezier patch or NURBS taking the form of Eq. (6). Let $\vec{r}$ denote a space point. Then, the square of the distance between $\vec{R}$ and $\vec{r}$ is given by

$$d = \left[\vec{R}(p,s,t) - \vec{r}\right] \cdot \left[\vec{R}(p,s,t) - \vec{r}\right] \tag{16}$$

The projection of $\vec{r}$ is defined to be the point $\vec{R}$ that minimizes $d$ of Eq. (16).

A local maximum or minimum distance is found by taking partial derivatives of $d$ with respect to $s$ and $t$ as follows:

$$\vec{R}_s \cdot \left[\vec{r} - \vec{R}\right] = 0 \equiv f(p,s,t) \tag{17}$$

$$\vec{R}_t \cdot \left[\vec{r} - \vec{R}\right] = 0 \equiv g(p,s,t) \tag{18}$$

Here, the subscripts $s$ and $t$ denote the derivatives with respect to $s$ and $t$, respectively. Applying Newton's iteration method, we obtain

$$f_s \Delta s + f_t \Delta t = -f \tag{19}$$

$$g_s \Delta s + g_t \Delta t = -g \tag{20}$$

where $\Delta s$ and $\Delta t$ are the changes in $s$ and $t$, respectively. Equations (19) and (20) are solved for $(\Delta s, \Delta t)$ at fixed $p$. $(s,t)$ are then updated as follows:

$$s^{Q+1} = \begin{cases} s^Q + [\omega(|\Delta s| - \epsilon) + \epsilon]\, sign(\Delta s) & \text{if } |\Delta s| > \epsilon \\ s^Q + \Delta s & \text{if } |\Delta s| \leq \epsilon \end{cases} \tag{21}$$

$$t^{Q+1} = \begin{cases} t^Q + [\omega(|\Delta t| - \epsilon) + \epsilon]\, sign(\Delta t) & \text{if } |\Delta t| > \epsilon \\ t^Q + \Delta t & \text{if } |\Delta t| \leq \epsilon \end{cases} \tag{22}$$

Here, $\omega$ is the relaxation factor, $\epsilon$ (typically, 0.02) is the cut-off parameter and the superscript $Q$ denotes the $Q\text{-}th$ iteration.

During iteration, the patch number $p$ is treated as a parameter and updated only if $(s,t)$ is outside the active region of patch $p$. When the calculated $(s,t)$ is outside the active region, the patch boundary data and the trim curve boundary data (if any) are used to find the adjacent patch to update $p$ and $(s,t)$ for the next iteration. In the case of patch boundary, Eq. (13) is used. In the case of trim curve, Eq. (15) is used to compute $\sigma_b$ on the trim curve of the adjacent patch. The physical space point computed at $\sigma_b$ on patch $p$ is projected onto the trim curve on the adjacent patch using the computed $\sigma_b$ as the initial guess. The corresponding $(s,t)$ on the adjacent patch is determined by Eq. (10). Then the computed $(s,t)$ is used to continue the iteration.

## Projecting A Point On A Curve And Distance Between Curves

Consider two parametrically defined curves, $\vec{r}(t)$ and $\vec{\xi}(\tau)$, each taking the form of Eq. (1). The square of the distance between $\vec{r}(t)$ and $\vec{\xi}(\tau)$ is given by

$$d = \left[\vec{r}(t) - \vec{\xi}(\tau)\right] \cdot \left[\vec{r}(t) - \vec{\xi}(\tau)\right] \tag{23}$$

At any point $\vec{\xi}(\tau)$ on the second curve we define the separation between the curves as the minimum distance d of Eq. (23). This is found by taking partial derivative with respect to $t$ while fixing $\tau$ as follows:

$$\vec{r}_t(t) \cdot \left[\vec{r}(t) - \vec{\xi}(\tau)\right] = 0 \equiv f(t,\tau) \tag{24}$$

Applying the Newton's iteration method, we obtain

$$f_t \Delta t = -f \tag{25}$$

or

$$\Delta t = -\frac{\vec{r}_t \left[\vec{r} - \vec{\xi}(\tau)\right]}{\vec{r}_t \cdot \vec{r}_t + \vec{r}_{tt} \cdot \left[\vec{r} - \vec{\xi}(\tau)\right]} \tag{26}$$

where $\Delta t$ is the change in $t$.

Along the curves there are a series of extrema where the curves are locally separated by a maximum or minimum distance. At these points, the partial derivative of Eq. (23) with respect to $\tau$ vanishes at fixed $t$ as follows:

$$\left[\vec{r}_\tau(t(\tau)) - \vec{\xi}_\tau(\tau)\right] \cdot \left[\vec{r}(t) - \vec{\xi}(\tau)\right] = 0 \equiv f(t(\tau), \tau) \tag{27}$$

where $t(\tau)$ is defined by the solution of Eq. (24) at fixed $\tau$.

Applying the Newton's iteration method, we obtain

$$f_\tau \Delta\tau = -f \tag{28}$$

or

$$\Delta\tau = \frac{-(\vec{r}_t t_\tau - \vec{\xi}_\tau) \cdot (\vec{r} - \vec{\xi})}{(\vec{r}_t t_{\tau\tau} + \vec{r}_{tt} t_\tau^2 - \vec{\xi}_{\tau\tau}) \cdot (\vec{r} - \vec{\xi}) + (\vec{r}_t t_\tau - \vec{\xi}_\tau) \cdot (\vec{r}_t t_\tau - \vec{\xi}_\tau)} \tag{29}$$

Here, $t_\tau$ and $t_{\tau\tau}$ may by evaluated from Eq. (24). For example, $t_\tau$ is evaluated as follows:

$$t_\tau = \frac{\vec{r}_t \cdot \vec{\xi}_\tau}{\vec{r}_{tt} \cdot (\vec{r} - \vec{\xi}) + \vec{r}_t \cdot \vec{r}_t} \tag{30}$$

So we can choose a $\tau$, iteratively solve for $t$ and $\vec{r}$ from Eq. (26), and then get $\Delta\tau$ from Eq. (29). This process continues until $\Delta\tau = 0$.

## Projecting A Point On A Surface Curve

Given a point in space $\vec{r}$ and a point on a parametrically defined curve, $\vec{R}(s(\tau), t(\tau))$, on a surface $\vec{R}(s, t)$, the square of the distance between $\vec{R}$ and $\vec{r}$ is given by

$$d = \left[\vec{R}[s(\tau), t(\tau)] - \vec{r}\right] \cdot \left[\vec{R}[s(\tau), t(\tau)] - \vec{r}\right] \tag{31}$$

A local maximum or minimum distance is found by taking partial derivative with respect to $\tau$ as follows:

$$\vec{R}_\tau \cdot \left[\vec{R} - \vec{r}\right] = 0 \equiv f(\tau) \tag{32}$$

Applying Newton's iteration method, we obtain

$$f_\tau \Delta\tau = -f \tag{33}$$

or

$$\Delta\tau = -\frac{\vec{R}_\tau \cdot (\vec{R} - \vec{r})}{\vec{R}_\tau \cdot \vec{R}_\tau + \vec{R}_{\tau\tau} \cdot (\vec{R} - \vec{r})} \tag{34}$$

where

$$\vec{R}_\tau = \vec{R}_s s_\tau + \vec{R}_t t_\tau \tag{35}$$

$$\vec{R}_{\tau\tau} = \vec{R}_s s_{\tau\tau} + \vec{R}_t t_{\tau\tau} + [\vec{R}_{ss} s_\tau + \vec{R}_{st} t_\tau] s_\tau + [\vec{R}_{ts} s_\tau + \vec{R}_{tt} t_\tau] t_\tau \tag{36}$$

## Curve And Surface Intersections

Consider a curve $\vec{C}(u)$, and a surface $\vec{R}(s,t)$. At an intersection between $\vec{C}$ and $\vec{R}$ we have

$$\vec{C}(u) - \vec{R}(s,t) = 0 \equiv \vec{f}(u,s,t) \tag{37}$$

constituting 3 equations in 3 unknowns, $u, s$, and $t$. Following the Newton's iteration method, we derive an iterative process as follows:

$$-\vec{C}_u(u)\Delta u + \vec{R}_s(s,t)\Delta s + \vec{R}_t(s,t)\Delta t = \vec{C}(u) - \vec{R}(s,t) \tag{38}$$

Isolating terms with cross and dot products, we have 3 equations,

$$-\vec{R}_s(s,t) \cdot \left[\vec{R}_t(s,t) \times \vec{C}_u(u)\right] \Delta u = \vec{R}_s(s,t) \cdot \left\{\vec{R}_t(s,t) \times \left[\vec{C}(u) - \vec{R}(s,t)\right]\right\} \tag{39}$$

$$\vec{C}_u(u) \cdot \left[\vec{R}_t(s,t) \times \vec{R}_s(s,t)\right] \Delta s = \vec{C}_u(u) \cdot \left\{\vec{R}_t(s,t) \times \left[\vec{C}(u) - \vec{R}(s,t)\right]\right\} \tag{40}$$

$$\vec{C}_u(u) \cdot \left[\vec{R}_s(s,t) \times \vec{R}_t(s,t)\right] \Delta t = \vec{C}_u(u) \cdot \left\{\vec{R}_s(s,t) \times \left[\vec{C}(u) - \vec{R}(s,t)\right]\right\} \tag{41}$$

Solving these equations, we have the iterative process,

$$\Delta u = \frac{\vec{R}_s \cdot \left[\vec{R}_t \times \vec{f}\right]}{\vec{C}_u \cdot \vec{n}}, \quad \Delta s = \frac{\vec{C}_u \cdot \left[\vec{R}_t \times \vec{f}\right]}{\vec{C}_u \cdot \vec{n}}, \quad \Delta t = \frac{\vec{C}_u \cdot \left[\vec{R}_s \times \vec{f}\right]}{\vec{C}_u \cdot \vec{n}} \tag{42}$$

since

$$\vec{C}_u \cdot \left[\vec{R}_s \times \vec{R}_t\right] = -\vec{C}_u \cdot \left[\vec{R}_t \times \vec{R}_s\right] = \vec{R}_s \cdot \left[\vec{R}_t \times \vec{C}_u\right] = \vec{C}_u \cdot \vec{n} \tag{43}$$

where $\vec{n} = \vec{R}_s \times \vec{R}_t$ is the surface normal at $(s,t)$, and $\vec{f} = \vec{C} - \vec{R}$. For a quilted (trimmed) surface, patch number $p$ will enter the iterative process and play the same role as that in section "Projecting A Point On A (Trimmed) Surface."

## Surface And Surface Intersections

When surface intersections are not given by trim curves, an intersection must be computed. Two surfaces generally intersect along a curve or curves in space. Initial points on any intersection curves may be found by creating a mesh of curves, usually constant parameter curves, on one of the surfaces, and then finding the intersection of these curves with the other surface. The mesh must be of sufficient density to find at least one point on all curves of intersection, but not too dense and causing an inordinate number of computations. When a point on any intersection curve is found, further intersection points are easily created along this curve. This curve may then be described parametrically by fitting an interpolating function through the generated points. This procedure is followed for each curve of intersection.

Consider two surfaces given by the equations

$$\vec{R}_1 = \vec{P}(s,t) \tag{44}$$

$$\vec{R}_2 = \vec{Q}(u,v) \tag{45}$$

each taking the form of Eq. (6). At an intersection point, we have 3 equations in 4 unknowns $s, t, u$, and $v$.

$$\vec{P}(s,t) - \vec{Q}(u,v) = 0 \equiv \vec{f}(s,t,u,v) \tag{46}$$

Following Newton's iteration method, we have

$$\vec{P}_s(s,t)\Delta s + \vec{P}_t(s,t)\Delta t - \vec{Q}_u(u,v)\Delta u - \vec{Q}_v(u,v)\Delta v = -\vec{f} \tag{47}$$

Since the system is overspecified, we need another equation for a unique solution.

Holding one of the paramters constant reduces the problem to a curve and surface intersection. Holding a parameter constant is also useful for finding a point on a particular intersection curve, such as the patch boundary. We could also hold the parameter constant which maximizes the denominator of Eq. (42) for the curve and surface intersection. Another way is to approach perpendicularly to the intersection curve along one of the surfaces. The tangent vector to the intersection curve $\vec{\tau}$, is given by,

$$\vec{\tau} = \vec{n}_P \times \vec{n}_Q \tag{48}$$

where $\vec{n}_P$ and $\vec{n}_Q$ are the local normals at $(s,t)$ and $(u,v)$, respectively,

$$\vec{n}_P = \vec{P}_s \times \vec{P}_t, \quad \vec{n}_Q = \vec{Q}_u \times \vec{Q}_v \tag{49}$$

The movement vector on surface $\vec{P}$ is given by $\vec{P}_s\Delta s + \vec{P}_t\Delta t$. The trace of the movement may be written in parameterized form as follows:

$$\vec{C}_s\Delta s = \vec{P}_s\Delta s + \vec{P}_t\Delta t \tag{50}$$

with the direction of movement constrained to the normal of the intersection curve, i.e.,

$$\vec{\tau} \cdot (\vec{P}_s\Delta s + \vec{P}_t\Delta t) = 0 \tag{51}$$

Now, the problem has been reduced to the intersection of curve $\vec{C}(s)$ and surface $\vec{Q}(u,v)$. Following section "Curve And Surface Intersection" and from Eq. (42), we have

$$\Delta s = -\frac{\vec{Q}_u \cdot \left[\vec{Q}_v \times \vec{f}\right]}{\vec{C}_s \cdot \vec{n}_Q}, \quad \Delta u = -\frac{\vec{C}_s \cdot \left[\vec{Q}_v \times \vec{f}\right]}{\vec{C}_s \cdot \vec{n}_Q}, \quad \Delta v = \frac{\vec{C}_s \cdot \left[\vec{Q}_u \times \vec{f}\right]}{\vec{C}_s \cdot \vec{n}_Q} \tag{52}$$

Here, $\vec{C}_s$ may be calculated from Eqs. (50)-(51) as follows:

$$\begin{aligned} (\vec{\tau} \cdot \vec{P}_t)\vec{C}_s\Delta s &= (\vec{\tau} \cdot \vec{P}_t)(\vec{P}_s\Delta s + \vec{P}_t\Delta t) \\ &= \left[(\vec{\tau} \cdot \vec{P}_t)\vec{P}_s - (\vec{\tau} \cdot \vec{P}_s)\vec{P}_t\right]\Delta s \qquad (53) \\ (\vec{\tau} \cdot \vec{P}_t)\vec{P}_s - (\vec{\tau} \cdot \vec{P}_s)\vec{P}_t &= [(\vec{n}_P \times \vec{n}_Q) \times \vec{n}_P]\Delta s \qquad (54) \end{aligned}$$

Since

$$(\vec{\tau} \cdot \vec{P}_t)\vec{C}_s \cdot \vec{n}_Q = [(\vec{n}_P \times \vec{n}_Q) \times \vec{n}_P] \cdot \vec{n}_Q = (\vec{n}_P \times \vec{n}_Q) \cdot (\vec{n}_P \times \vec{n}_Q) \tag{55}$$

we cannot have a nonzero denominator unless $\vec{n}_P \times \vec{n}_Q = 0$, which means the surfaces are locally parallel and there is no intersection anyway. Substituting in the above we get the iteration process,

$$\begin{aligned} \Delta s &= -\frac{\left[(\vec{n}_P \times \vec{n}_Q) \cdot \vec{P}_t\right]\left[\vec{Q}_v \times \vec{f}\right] \cdot \vec{Q}_u}{(\vec{n}_P \times \vec{n}_Q) \cdot (\vec{n}_P \times \vec{n}_Q)} \qquad (56) \\ \Delta t &= \frac{\left[(\vec{n}_P \times \vec{n}_Q) \cdot \vec{P}_s\right]\left[\vec{Q}_v \times \vec{f}\right] \cdot \vec{Q}_u}{(\vec{n}_P \times \vec{n}_Q) \cdot (\vec{n}_P \times \vec{n}_Q)} \qquad (57) \\ \Delta u &= -\frac{[(\vec{n}_P \times \vec{n}_Q) \times \vec{n}_P] \cdot \left[\vec{Q}_v \times \vec{f}\right]}{(\vec{n}_P \times \vec{n}_Q) \cdot (\vec{n}_P \times \vec{n}_Q)} \qquad (58) \\ \Delta v &= \frac{[(\vec{n}_P \times \vec{n}_Q) \times \vec{n}_P] \cdot \left[\vec{Q}_u \times \vec{f}\right]}{(\vec{n}_P \times \vec{n}_Q) \cdot (\vec{n}_P \times \vec{n}_Q)} \qquad (59) \end{aligned}$$

As previously defined, $\vec{n}_P$ and $\vec{n}_Q$ are in the direction of the local normals.

$$\vec{n}_P = \vec{P}_s \times \vec{P}_t, \quad \vec{n}_Q = \vec{Q}_u \times \vec{Q}_v, \quad \vec{f} = \vec{P}(s,t) - \vec{Q}(u,v) \tag{60}$$

When the convergence is satisfied, the next point may be found by marching along the surfaces in the direction of the tangent vector $\vec{\tau} = \vec{n}_P \times \vec{n}_Q$, and iterating again. Along the tangent vector,

$$\Delta\vec{R} = \vec{P}_s\Delta s + \vec{P}_t\Delta t = \frac{\vec{\tau}}{|\vec{\tau}|}\Delta\sigma \tag{61}$$

or

$$\Delta\vec{R} = \vec{Q}_u\Delta u + \vec{Q}_v\Delta v = \frac{\vec{\tau}}{|\vec{\tau}|}\Delta\sigma \tag{62}$$

where $\sigma$ is the arc length. Then $\Delta s$ and $\Delta t$ along the intersection curve may be calculated by the vector cross product of Eq. (61) and $\vec{P}_s$ and $\vec{P}_t$, respectively,

$$-(\vec{P}_s \times \vec{P}_t)\Delta s = \vec{P}_t \times \left[(\vec{P}_s \times \vec{P}_t) \times \vec{n}_Q\right] \frac{\Delta\sigma}{|\vec{\tau}|} = (\vec{P}_s \times \vec{P}_t)(\vec{P}_t \cdot \vec{n}_Q)\frac{\Delta\sigma}{|\vec{\tau}|} \tag{63}$$

$$\Delta s = -(\vec{P}_t \cdot \vec{n}_Q)\frac{\Delta\sigma}{|\vec{\tau}|} \tag{64}$$

$$\Delta t = (\vec{P}_s \cdot \vec{n}_Q)\frac{\Delta\sigma}{|\vec{\tau}|} \tag{65}$$

Similarly, $\Delta u$ and $\Delta v$ are obtained by the vector cross product of Eq. (62) and $\vec{Q}_u$ and $\vec{Q}_v$, respectively,

$$\Delta u = (\vec{Q}_v \cdot \vec{n}_P)\frac{\Delta\sigma}{|\vec{\tau}|} \tag{66}$$

$$\Delta v = -(\vec{Q}_u \cdot \vec{n}_P)\frac{\Delta\sigma}{|\vec{\tau}|} \tag{67}$$

Here, $\Delta\vec{\tau}$ is computed as follows:

$$\begin{aligned} \Delta\vec{\tau} &= \vec{\tau}_s\Delta s + \vec{\tau}_t\Delta t + \vec{\tau}_u\Delta u + \vec{\tau}_v\Delta v \\ &= \left[-\vec{\tau}_s(\vec{P}_t \cdot \vec{n}_Q) + \vec{\tau}_t(\vec{P}_s \cdot \vec{n}_Q) + \vec{\tau}_u(\vec{P}_v \cdot \vec{n}_P) - \vec{\tau}_v(\vec{P}_u \cdot \vec{n}_P)\right] \frac{\Delta\sigma}{|\vec{\tau}|} \end{aligned} \tag{68}$$

### Surface And Plane Intersections

For the user's convenience, a space plane is specified by a set of three non-colinear points or by a reference point and a normal to the plane. Intersection between a quilted surface and the user-specified plane is done exactly in the same manner as that described in section "Surface And Surface Intersections" once the plane is represented by a quilt containing a single patch.

## SURFACE GRID GENERATION

Both structured and unstructured grid generators used in RAGGS are briefly described in this section. References will be cited for readers who are interested in details of the numerical algorithms actually implemented for the surface grid generations.

TFI Method.—The transfinite interpolation (TFI) of Ref. 2 is coupled with the space point projection, Eqs. (16)-(22), to determine the interior grids of a 4-sided block. A subsection approach is employed. The approach allows a 4-sided block grid be calculated subsection by subsection. Each subsection is defined by its two opposite diagonal points and can easily be specified interactively by point-and-click. A subsection grid is computed in two steps. First, the 4-sided subsection grid is calculated by TFI based on the sectional edge grids. Second, Eqs. (19)-(20) are used to project the TFI grids onto the quilted surface. At each point, a set of initial guesses $(p, s, t)$ is formed by gathering all the available values at immediately adjacent grid points. Additional sets of $(p, s, t)$ may also be specified by user. The solution is taken to be the one that has the minimum distance among all the calculated solutions.

When a grid is generated on a single patch, TFI may be optionally applied to the surface parametric variables. In this case, the grid position vectors are evaluated directly from the patch considered using the TFI interpolated values of parametric variables.

PDE Method.—The elliptic grid generation system on arbitrary curved space surface can be written, in tensor form[3], as follows:

$$\sum_{m=1}^{2}\sum_{n=1}^{2} g g^{mn} \vec{r}_{\xi^m \xi^n} + \sum_{n=1}^{2} g g^{nn} P_n \vec{r}_{\xi^n} - g H \vec{n} = 0 \tag{69}$$

Here $\vec{r}$ is the grid position vector, $\xi^i$ are the curvilinear coordinates, $P_n$ are the grid control functions, $g^{il}$ is the contravariant metric tensor, and $g = det(g_{ij})$ with $g_{ij}$ the covariant metric tensor. $H$ is the twice of the surface mean curvature and $\vec{n}$ is the unit surface normal. Equation (69) is solved numerically using the relaxation methods described in Ref. 4 once the differential operators are replaced by central-difference operators. Equations (19)-(20) are used to snap the solution grid points onto the quilted surface at the end of each iteration.

The grid control functions $P_n$ are evaluated at the 4-sided block edges and the boundary values are interpolated into the field. Both Thomas/Thomas-Middlecoff's method[5,6] and Sorenson's method[7,8] have been implemented for the construction of orthogonal boundary grids. However, Sorenson's method is implemented based on Thompson's method of Ref. 2 with the following two modifications. One, the central-difference operator is used to approximate the second-order derivatives normal to the boundary instead of the one-sided difference operator. Second, the grid control functions are updated immediately after the contributions from a given orthogonal boundary section have been evaluated and the control function increments are evaluated against the most current values. The first modification provides a consistent numerical method in the sense that both the grid position vector and control functions satisfy the same governing equation as the solution converges, and a grid initially orthogoanl will remain and be returned as the solution. The second modification has provided a more stable iterative process.

Similarity Method.—An existing grid may be fine-tuned to have a similar or identical distribution in a specified base variable, the arc length or one of the three Cartesian coordinates, along a specified coordinate direction. To accomplish this, the edge distributions of the specified base variable are interpolated into the field. Then, the grid points are re-evaluated line by line in the specified coordinate direction and projected onto the quilted surface. This can provide a simple means of achieving a cylindrical-coordinate-like grid. Also, it can provide panels on an aircraft configuration with constant spanwise or axial sections as required by some panel codes.

Marching Method.—A grid may be constructed on a quilted surface from a given line grid (open or

closed) by marching in the direction normal to the line segment with a specified step size. The marching direction is the direction of the vector formed as the cross product (in the order specified) of the surface normal and the tangent vector to the line segment.

Smoothing.—An existing grid may be smoothed by replacing each of the interior points by the averaged value of its adjacent grid points. The number of passes of smoothing is specified by the user.

## Unstructured Surface Grid Generation

Details of our unstructured grid generation algorithm may be found in Ref. 9. Briefly, the technique is based closely on the Löhner-Parikh's[10] data structures and the 2-D advancing front triangulation scheme described in Ref. 11. However, the implementation is made in the 3-D physical space coupled with the space point projection, Eqs. (16)-(22). Interactively, a grid generation starts with a rectangular octree box specified just big enough to cover the entire region to be triangulated. The advancing front is constructed automatically as the user is pointing and clicking line segments on the screen. During triangulation, every new nodal point is introduced to form a nearly equilateral triangle and satisfy the mesh consistency requirement with a base selected from the front. Equations (19)-(20) are used to iteratively locate the new nodal point on the quilted surface.

Since the unstructured grid generation is based on the same geometry database as that of the structured grid generation, geometry manipulators and domain blocking tools developed for structured grid generation can be used directly. With this new capability, RAGGS provides a unique environment where both structured and unstructured grids can be generated and manipulated on a common geometry database.

## SAMPLE CASES

Sample cases are presented herein to demonstrate the capabilities of RAGGS in handling trimmed surfaces and generating grids on an unstructred quilt of patches and/or trimmed surfaces.

Figure 4 demonstrates the capability of RAGGS to display trimmed surfaces. Shown in the figure is a portion of an impeller. The geometry is defined by a series of trimmed surfaces. Figures 4-(a) and (b) show the shaded displays of the untrimmed and trimmed patches, respectively. One can hardly picture what the geometry would look like from the untrimmed surface display in Fig. 4-(a). But it is clearly shown in Fig. 4-(b). Figures 4-(c) and (d) show how the two patches corresponding to the front face and the right side of the geometry are discretized for display purposes. In RAGGS a structured grid is used to discretize the patch in the parametric space with the grid size adaptive to the local curvature in each direction of the parametric variable as shown in Fig. 4-(c). For a trimmed patch, the active region is automatically triangulated in parametric space, using the advancing front technique of Ref. 9, with the trim curves as the initial front. Figure 4-(d) shows the triangulation of the active regions of the two patches shown in Fig. 4-(c). It may be noted that in RAGGS the trimmed and untrimmed displays can easily be toggled on and off. Also noted is that no additional patch definition is given to the trimmed patch and the definition of the original untrimmed patch is used for the geometry.

Figure 5 demonstrates the capability of RAGGS for generating structured and unstructured grids on unstructured quilts of patches. Figure 5-(a) shows a B-1B forebody with fins. Four quilts are shown here: the upper and lower body quilts and the upper and lower fin quilts. No trim curves were provided so a surface intersection was computed between the body and fin quilts. The irregular arrangement of patches is clearly indicated by the wireframe display in Fig. 5-(a). The calculated structured and unstructured grids are shown in Figs. 5-(b) and (c), respectively.

Figure 6 demonstrates the capability of RAGGS for gridding on trimmed surfaces. Figure 6-(a) shows a plane and a bump (part of semi-spherical surface represented by NURBS). The four edge grids are specified entirely on the flat plane, and the interior grids are generated while considering the trimmed surfaces. As shown in Fig. 6-(b), the interior grid points have been properly positioned on the trimmed patches. It is noted that the unstrimmed surfaces are the only surfaces used to define the geometry.

## CONCLUSIONS

Capability of generating structured and unstructured grids directly on CAD generated geometry with/whitout trimmed surfaces has been described and demonstrated. Both geometry integrity and patch structure are retained from IGES file to CFD geometry database.

## REFERENCES


1. Woan, C-J, Clever, W. C., and Tam, C. K., "RAGGS: Rockwell Automated Grid Generation System," AIAA Paper AIAA-94-0206, AIAA 32nd Aerospace Sciences meeting & Exhibit, January 10-13, 1994.

2. Thompson, Joe. F. and Gatlin Boyd, "Program EAGLE User's Manual: Vol III-Grid Generation Code," AFATL-TR-88-117, Vol III, Airforce Armament Laboratory, Eglin Air Force Base, Florida, September 1988.

3. Thompson, Joe. F., Warsi, Z. U. A., and Mastin, C. Wayne, *NumericalGridGeneration*, North-Holland·New York·Oxford, 1985.

4. Woan, C-J, "Multi-Block Multigrid Calculations of a System of Elliptic Grid Generators," AIAA Paper 88-0312, AIAA 26th Aerospace Sciences Meeting, Reno, Nevada, January 11-14, 1988.

5. Thomas, P. D. and Middlecoff, J. F., "Direct Control of the grid Point Distribution in Meshes Generated by Elliptic Equations," AIAA Journal, Vol. 18, No. 6, pp. 652-656, June, 1980.

6. Thomas, P. D., "Composite Three-Dimensional Grids Generated by Elliptic Systems," AIAA Journal, Vol. 20, No. 9, pp. 1195-1202, September 1982.

7. Sorenson, Reese L., "A Computer Program to Generate Two-Dimensional Grids about Airfoils and Other Shapes by Use of Poisson's Equation," NASA TM-81198, May 1980.

8. Sorenson, Reese L., "Three-Dimensional Elliptic Grid Generation about Fighter Aircraft for Zonal Finite-Difference Computations," AIAA Paper 86-0429, AIAA 24th Aerospace Sciences Meeting, Reno, Nevada, January 6-9, 1986.

9. Woan, C-J, "Unstructured Surface Grid Generation on Unstructured Quilt of Patches," AIAA Paper AIAA-95-2202, AIAA 26th Fluid Dynamics Conference, June 19-22, 1995.

10. Löhner, R. and Parikh, P., "Generation of Three-Dimensional Unstructured Grids by the Advancing-Front Method," AIAA Paper AIAA-88-0515, AIAA 26th Aerospace Sciences Meeting, January 11-14, 1988.

11. Peraire, J., Peiro, J., Formaggia, L., Morgan, K., and Zienkiewicz, O. C., "Finite Element Euler Computation in Three Dimensions," AIAA Paper AIAA-88-0032, AIAA 26th Aerospace Sciences Meeting, January 11-14, 1988.

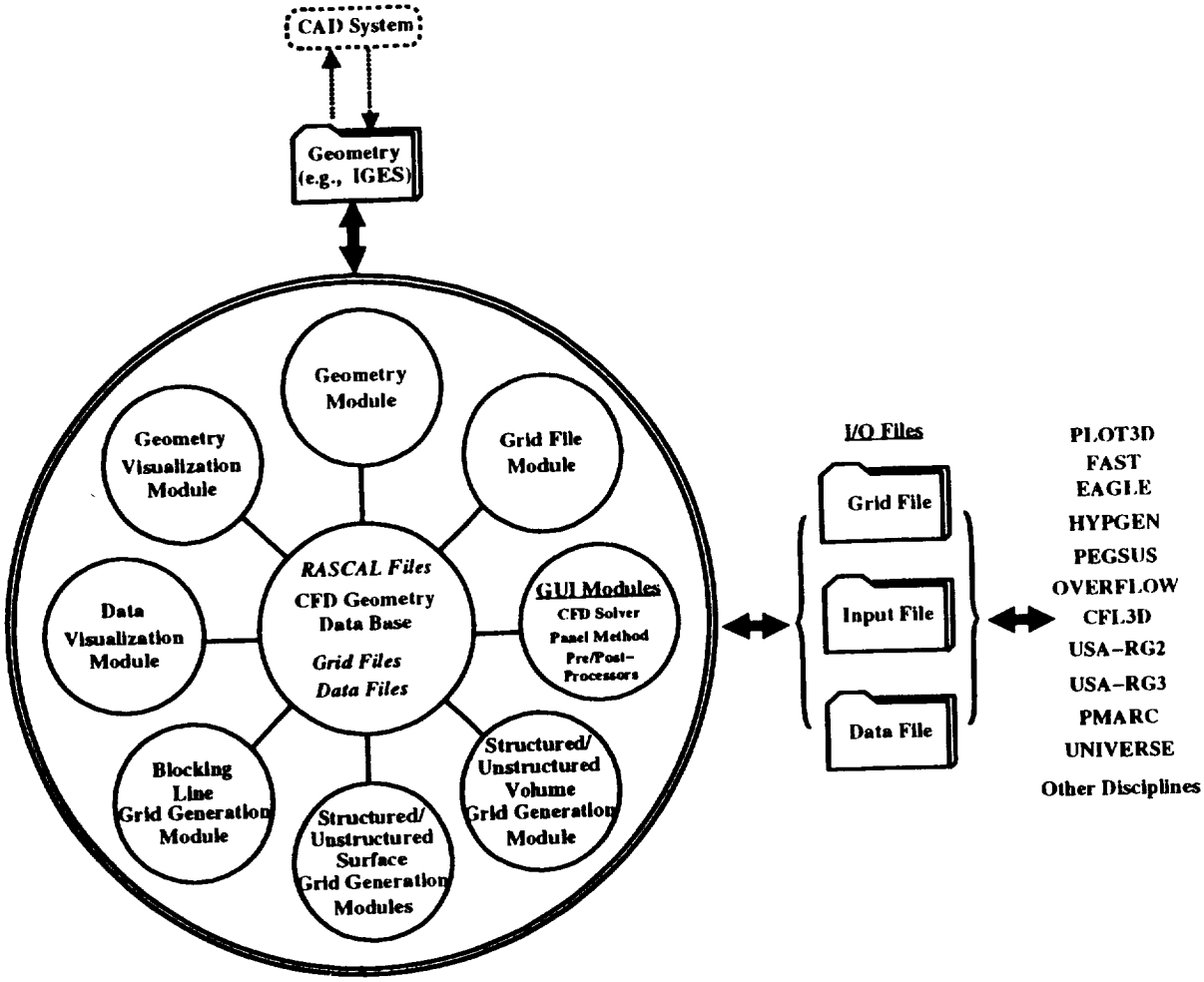



Figure 1.–The program structure of RAGGS.

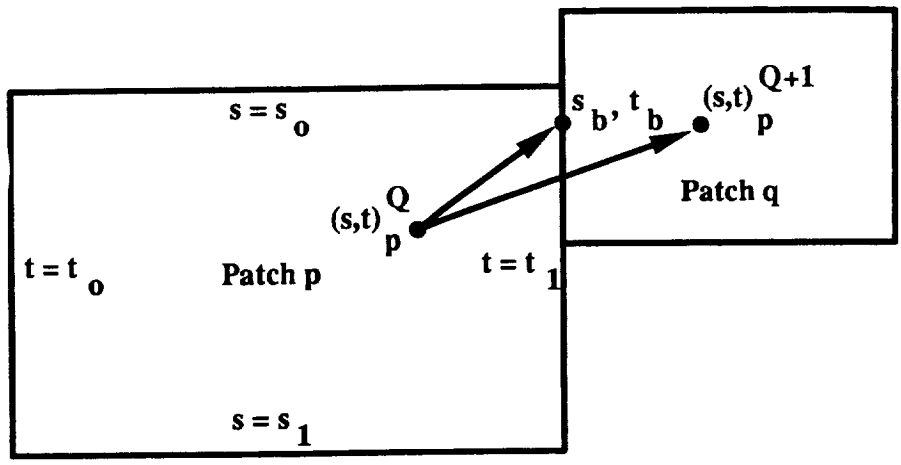



Figure 2.–Typical boundary connectivity between two adjacent patches and determination of new patch and parametric variables during iteration.

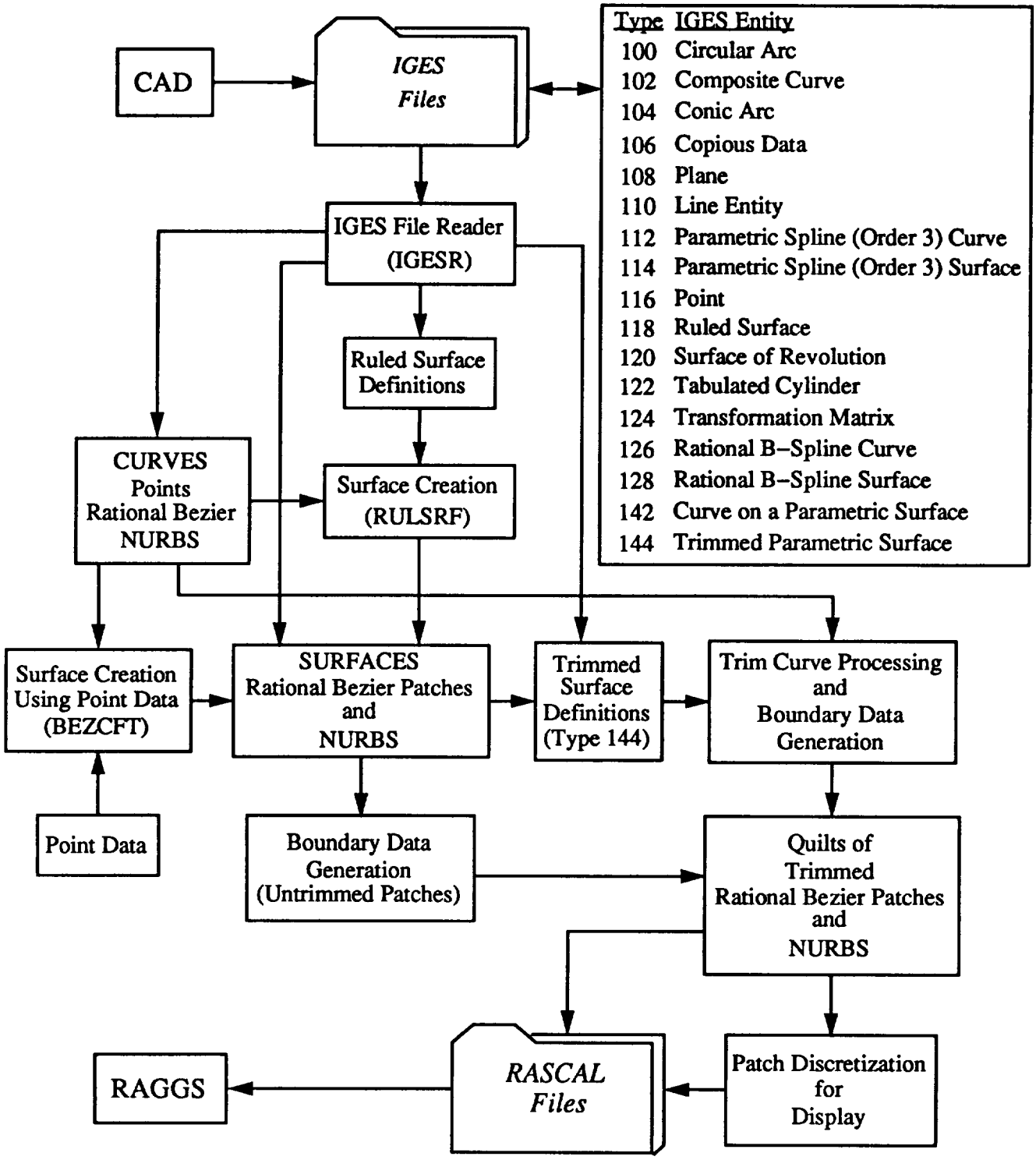
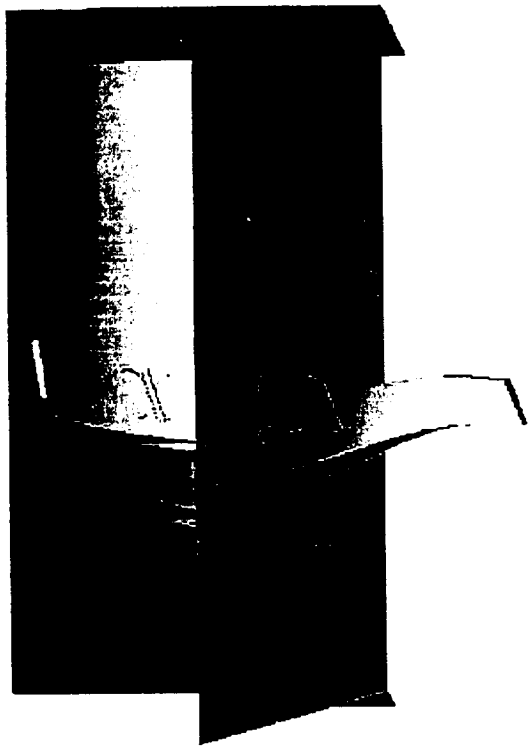
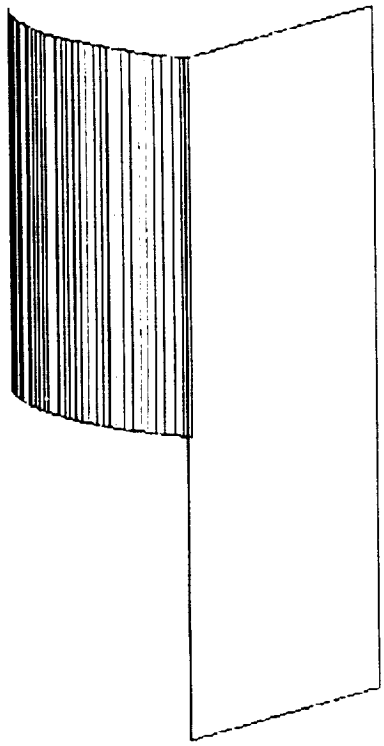


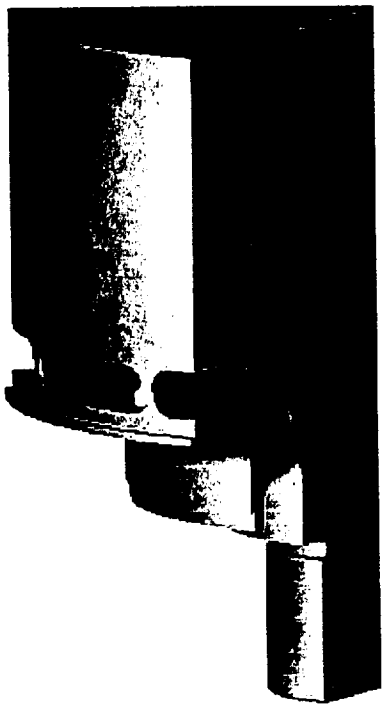

Figure 3.–The IGES/RAGGS interface.



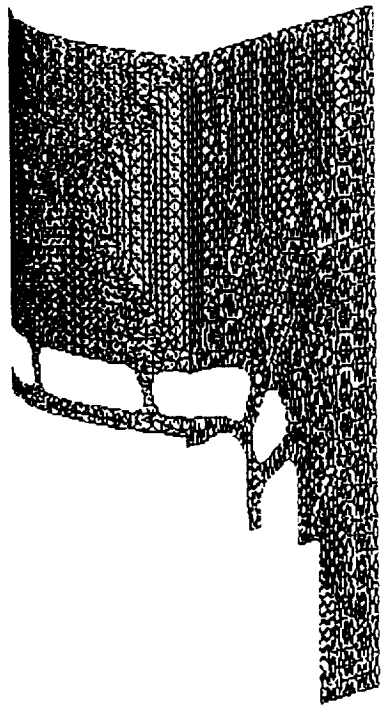
(a) Shaded display of untrimmed surfaces



(c) Structured Polygons for untrimmed patches

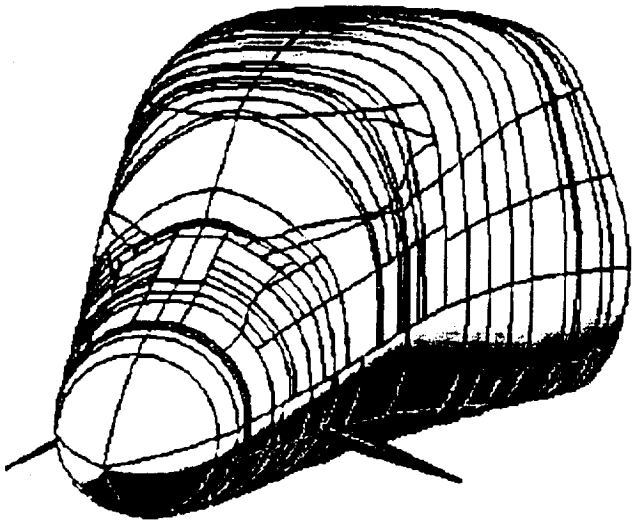


(b) Shaded display of trimmed surfaces

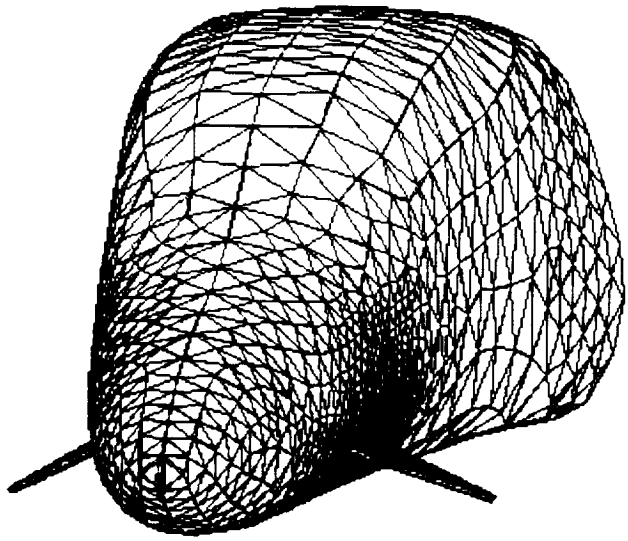


(d) Triangulation of trimmed patches

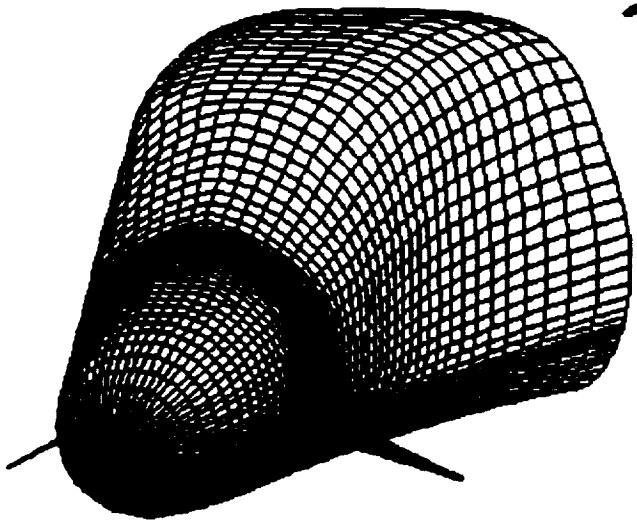
Figure 4.–Capability of displaying trimmed and untrimmed surfaces.



(a) Bezier patch–defined B–1b forebody showing unstructured quilt of patches

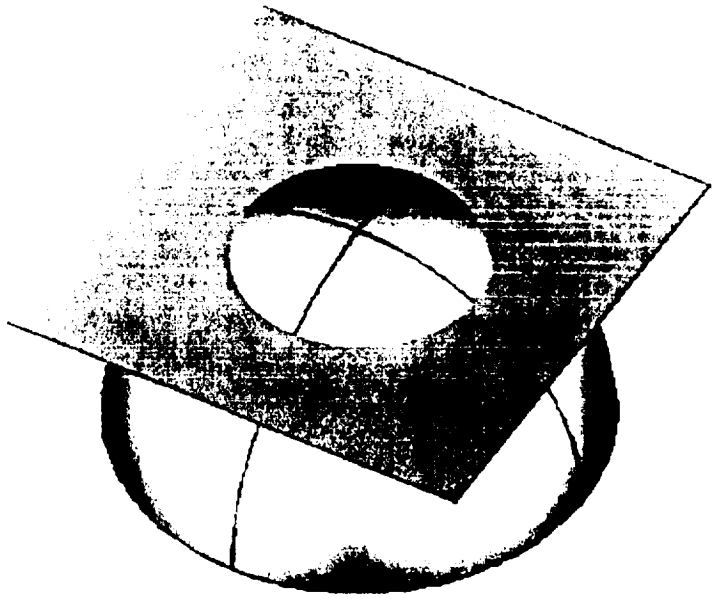


(c) Unstructured surface grid

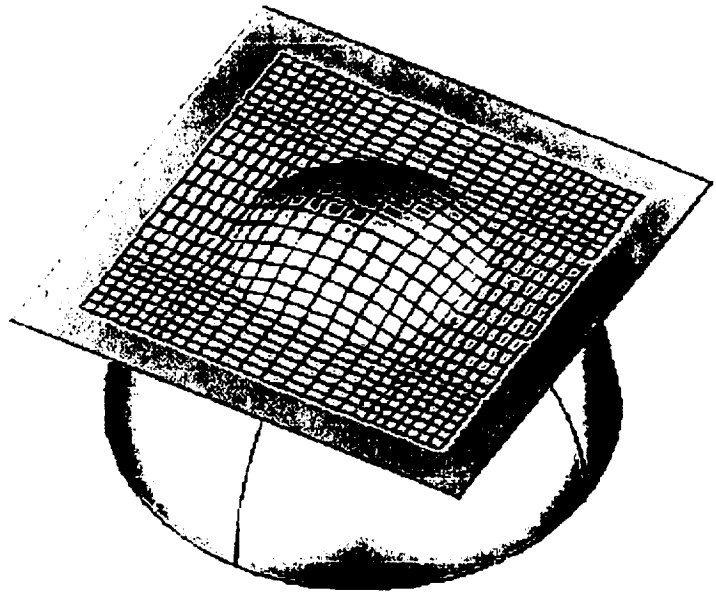


(b) Structured surface grid

Figure 5.–Capability of structured and unstructured grid generations on unstructured quilt of patches.



(a) Patch–defined plane and bump



(b) A structured grid on the trimmed surface

Figure 6.–Capability of gridding on trimmed surfaces.

N95-28756

# GEOMETRIC GRID GENERATION

David Ives
Pratt & Whitney
East Hartford, CT 06108

## ABSTRACT

This paper presents a highly automated hexahedral grid generator based on extensive geometrical and solid modeling operations developed in response to a vision of a designer-driven **one day** turnaround CFD process which implies a designer-driven **one hour** grid generation process.

## THE VISION

To affect the everyday aerodynamic design processes, a CFD system must be capable of generating the grid, setting the flow boundary conditions, solving for the flow, and completing the graphics and performance post processing in less than one 20 hour period, leaving at least 4 hours for the designer to conceive and define a new geometry to improve on the current design. This new design may have a different topology than previous cases. The flow solver unattended running occupies 16 hours of this cycle. The 24 hour design-analysis cycle seems to be natural human cycle, and allows a large number of design iterations. Each of the CFD system components must be sufficiently automated to allow the designer to operate them with minimal specialized training and support from CFD "experts". This one day turnaround is also described in Reference 1 which states: "In aerodynamic design work for commercial aircraft, CFD calculation results for aerodynamic analysis should be produced ... with short turnaround times (for large, complex calculations, **1 day**, say)". Likewise, Reference 2 states "The significance of this is that the total elapsed time for redesigning and arriving at a wing which meets all aerodynamic and manufacturability requirements can now be envisioned to approach the order of **one day**!"

These requirements break down to allowing an hour for high level automated grid generation, which is the focus of this paper. The high automation level reduces variation in grids generated by different users. With this constraint in mind, a grid generation system was developed which includes **only** technologies compatible with the **one hour** limit. This approach essentially starts from the desired market-driven goal and works backward to select the technologies used, rather than making continuous incremental improvements to current technology to work forward toward the one hour requirement. This is a high-risk, high-payoff approach. The initial demonstration described below is for Euler flows only; similar viscous flow grid technology is currently being developed. This approach can also be used with conventionally generated grids to quickly add geometric details.

## PROCESS OVERVIEW

The geometric grid generation process used here typically starts with a single component body-conforming base grid from a conventional grid generator, as illustrated in Figure 1. A surface defining additional geometry to be modeled, shown in Figure 2, is then embedded in the base grid. The first embedding step is to close the surface as in Figure 3. Each base grid cell is then examined to determine if it is inside or outside the closed surface; if outside, it is retained and if inside it is marked as solid (inactive in the CFD flow solver). The process at this stage is illustrated in Figure 4 where the embedded surface is represented as a stairstep. This stairstep surface is then distorted by moving all nodes to the surface of Figure 2. A smoothing is coupled with this process; the resulting hexahedral grid in Figure 5 conforms to the body. The following examines each of these steps in detail.

## INPUT PROCESSING

The geometric grid generation process assumes that a valid boundary representation (B-rep) solid model which completely specifies the space to be gridded has been produced by the CAD system used by the designer.The B-rep solid model surface is received from the CAD system as a large number of small triangles. The most popular format for CAD output in this form is a sterolithography file, which is a simple universal standard (in practice) currently supported by most major CAD systems. This allows us to use our current CAD system to process IGES input. The NASA IGES and NGP translators of References 3 and 4 may also be used to provide a separate IGES interface.

In practice, the quality of these solid models varies with the quality of the CAD system, and some clean-up operations are currently necessary. As CAD systems mature, the need for these separate clean-up operations will decrease, being transferred to the CAD system itself. The current grid generation system supports the following high level clean-up and manipulation operations on the input triangle surface primarily in a UNIX command line controlled mode:

1) interactively select surfaces/patches of object to be gridded
2) cap open surfaces under interactive user control:
   a) cap single open or closed curve
   b) cap region between 2 closed curves
3) remove small details deemed unimportant by using capabilities 1 & 2
4) topological analysis of input surfaces to verify B-rep solid model integrity
5) merge points differing within a tolerance
6) reflect surfaces about symmetry planes
7) extend open surfaces to resolve intersection tolerances
8) Boolean trim, union,. etc. of objects - i.e. trim a pylon surface with wing and nacelle surfaces, then union them to obtain a single wing/pylon/nacelle surface

Please note that most of the above operations are very high level, usually requiring at most a single number to describe the operation, and thus are amenable to command-line control. The exceptions are options 1-3 which require a visualization and selection of the different surfaces and patches. The SUN "leotool" visualization system is used in a separate synchronous UNIX process to select the surfaces and patches. All that is required is to write a visualization file in the Wavefront ".obj" format which separates the surface/patch components into groups. These groups can be toggled on and off in leotool, allowing each group to be identified for control of the surface/patch selection process.

Another way to receive a B-rep solid model is to read a VSAERO panel method input file of Reference 5, which can be produced by other means such as the ICEM system of Reference 6. Such surface models tend to be of high quality and require only very modest or no clean-up to achieve a perfect solid model surface specification.

The basic philosophy is to let the CAD system or other tools which the designer uses on a daily basis specify the boundaries of the space to be gridded. This approach reduces training and overall complexity as the designer does not need to learn two different CAD systems, one for everyday work and the other for CFD only. It is noted that the resulting grids are patch-independent; they depend only on the B-rep surface as defined by the triangulation used to create the sterolithography file and not on the B-rep patch edge locations unless the user specifies specific patch edges to retain in the final grid.

Base grid. - The flow region is determined by **embedding** the input surface in a base grid, which can

range from a simple uniform Cartesian grid composed of a rectangular collection of cubes to a grid generated by a current interactive grid generator such as ICEM, NGP, or others.The embedding has three steps: automatic blocking, surface snapping, and quality improvement. These steps are discussed below. For purposes of this discussion a "solid" point, edge, face, or cell lies in a region in which there is no flow, and a "fluid" point, edge, face, or cell lies in a region where there is flow. A "computational space smoothing" refers to moving a point to the average position of all points linked to the point. A "physical space smoothing" refers to moving a point to satisfy a Laplace equation on a stencil involving all points linked to the point and requires knowledge of which finite difference stencil points are opposite each other as described in Reference 7.

## AUTOMATIC BLOCKING

The geometric grid generator automatically determines the required blocking in a batch workstation run on using UNIX command-line options to select the desired blocking rules. The blocking is done on a cell-by-cell basis, which is simple for a computer, rather than blocking on large rectangular collections of cells as is done in interactive grid generators. The normal blocking rule used here is a simple local geometric test based on inside/outside tests of the 8 vertices of each cell and on intersection tests for 12 edges of each cell. This blocking rule is:

1) if 5 or more vertices are inside the surface, then the cell is solid
2) if 4 vertices are inside the surface and the average of the 8 vertices is inside the surface, then the cell is solid
3) if 4 or more edges intersect the surface ,then the cell is solid
4) otherwise the cell is fluid

The intersection tests in the blocking rule allow the correct detection of thin surfaces such as the trailing edge region of wings. A more sophisticated blocking rule could count intersections along each edge. If there are two intersections and both end points of the segment are in a fluid region then the edge is solid, while if both end points of the segment are in a solid region then the edge is fluid. These edge states could be included in determining if the cell is solid or fluid. This would allow the detection of cracks as well as thin trailing edges.

The inside/outside tests are based on ray tracing as described in Reference 8. If a ray from the point being tested to positive infinity in the Z-direction intersects the surface an odd number of times then the point is inside, otherwise it is outside. In practice, rays are sent in two opposite directions and the code checks to verify that both rays agree on the containment. If they do not agree, then a new direction is chosen and the process is repeated. For difficult cases (ie: cases where tolerances are nearly the same size as roundoff error) an option is included to require consensus among six orthogonal rays issued from the point to infinity. Logic is included to ensure that a ray hitting a triangle edge is given half an intersection count for each triangle independent of roundoff by the use of special symmetric coding. Of course, each edge is shared by exactly two triangles since we currently require two-manifold surfaces. A new direction is chosen if a ray hits a triangle vertex, although we could have included logic to give an intersection count of 1/N, where N is the number of edges sharing the vertex. The technology described here is equally applicable to Chimera grid generation, having much in common with the overlapping stencil generation described in Reference 9.

Handling objects which extend outside base grid. - There are cases in which the object to be gridded extends outside the base grid in some areas, and is inside or on the base grid surface elsewhere. For each stairstep cube with a quad face lying on an external block boundary (ie: the quad occurs only once), which we denote as a "candidate quad", we must decide if the face is to be retained in the

stairstep surface quad list. This is controlled by an overall control which instructs the system to either retain all of the candidate quads, or remove all of them. A local control, applied on selected base grid block faces, instructs the system to do the opposite of the overall control. In practice, the overall control is usually set to reject all candidate faces and the local control is applied to retain quads in a very few block faces.

Computational speed. - The key issue here is computational speed in determining the containment of typically one million points within, and the intersection of three million line segments with, a surface that may be composed of up to 100,000 triangles. Fortunately, the extensive literature of computer graphics, data structures, and searching contains efficient ray casting and searching algorithms (References 8, 10-13) which require only a few minutes of elapsed time on a workstation to perform these operations even for a curvilinear base grid. It is surprising that these speeds can be achieved, given the very large operation counts (3,000,000 times 100,000) that a straightforward approach would require. Techniques such as those in Reference 14 can often substantially reduce the number of triangles required to define a surface to a specified tolerance by selectively removing triangles in regions of low curvature, substituting a smaller number of large triangles for a large number of small triangles. This can help in the initial code development phase by making acceleration techniques less crucial at first. The methods used in the current work are based on sorting, classifying, and range box tests to preselect candidates for geometrical operations. It turns out that the time and storage required for these processes is only weakly (logarithmically) dependent on the number of triangles defining the input surface, so we can easily cope with 100,000 input surface triangles. Thus we have not yet added the capability to reduce the number of input surface triangles which became available after the current grid code development had progressed beyond the initial stages. However, it seems that this technique would be well suited to help clean up input surfaces which often have poorly distributed (both in size and in aspect ratio) triangles defining the surface.

## SURFACE SNAPPING

The next step is to specify the "goal surface" which is the portion of the input surface which lies inside the surface of the base grid. This is achieved by using the base grid outer surface, expressed as a collection of triangles, to clip (in a Boolean sense) the input surface, which is also expressed as a collection of triangles. This operation is necessary only if the input surface extends beyond the base grid surface, otherwise the goal surface and the input surface are identical. The surface shown in Figure 3 extends outside the base grid surface (ie: it extends inside the fan cowl.) The base grid outer surface is obtained by loading each of six faces of each base grid cell into an array, then removing those faces which occur twice in the array to leave only those (surface) faces occurring once in the array.

Association. - A geometric and topological analysis is then performed on the goal surface to identify corners, apexes, paths, circuits, and boundaries. A sketch describing these features is included in Figure 6. A corner is a point in the goal surface which must be represented as a point in the snapped surface. Sharp edges in the snapped surface must lie along edge paths in the goal surface. A sharp edge is formed when two adjacent input triangles meet at a shared edge with less than a user-specified included angle. A path is a collection of sharp edges. Paths may either stop on the surface (called a pendant path) or terminate at a corner. A corner is defined as the meeting of three or more sharp edges or the isolated end of a pendant path. An apex is the tip of a conical surface. A boundary is an intersection curve (if any) of the input surface and the base grid surface. A circuit is a collection of paths surrounding a surface. To retain geometric and topological fidelity, corners and apexes in the input surface must correspond directly to points in the generated grid. Likewise, paths and boundaries in the goal surface must correspond directly to interior edges and boundary edges in the

generated grid. For geometrical fidelity, it is undesirable to allow a sharp edge to be represented by the diagonal (rather than the edge) of one of the cells. The process of associating stairstep surface vertices with the matching input surface corners, apexes, paths, boundaries, and the surfaces enclosed by the circuits is called "association". The most difficult task in the current implementation of geometric grid generation is the automatic association of paths on the quad surface with the sharp edge paths on the input triangular surface. For base grids much finer than the path separations, this task is straightforward and reliable. It becomes difficult when the distance between paths is small compared to the base grid resolution.

Currently, the association is done as a three step process. In the first step, a "bead" is swept along the triangle path, and the quad surface points closest to the bead are collected. In the second step, adjacent duplicate quad points are culled and a connected path is built by linking points which are adjacent in the quad faces. Then the remaining gaps are connected up to a distance of three links (six links near the path ends). Then remaining gaps are connected across quad diagonals if necessary. All of this is done under the constraint that no existing quad path be crossed. In the third step,the quad path is improved by removing self-canceling segments and special handling is applied to the path ends where multiple paths may meet.A refined version of this method replaces the bead point by the closest intersection of the triangle surface normal at the bead point with the quad surface. This uses additional information (ie: the normal vector) to distinguish between possible quad paths which lie close to each other. This logic is essential at a wing/fuselage intersection.

The point associations are established first, with interior and boundary points on the goal surface (corners or apexes) being associated, respectively, with interior and boundary points on the stairstep surface. The association criteria is based on the shortest distance between points in the stairstep and goal surfaces which are eligible to associate. As soon as a point is associated, it is flagged as ineligible for further association. Then boundary edge curves in the input surface are associated with collections of edges in the stairstep surface by associating eligible stairstep vertices with the input paths. Interior edges are then associated between the two surfaces in a similar manner. Finally the areas contained within closed circuits established by collections of the goal surface sharp edge curves (which are in turn collections of edges which are in turn linked collections of points) are associated with vertices between the corresponding circuits in the stairstep surface. The result is a list of corresponding points, a list of corresponding curves matched with collections of stairstep vertices, and a list of corresponding areas associated with stairstep vertices. Corresponding paths for the surfaces of Figures 2 and 4 are illustrated in Figure 7.The vertices in the stairstep grid surface are then moved to their associated points by a process we call "snapping" (in analogy to the CAD meaning of snapping), or the the closest point on their corresponding curve, or to the closest point on their corresponding surface. The HEXAR grid generator from Cray Research described in Reference 15 also operates in this manner. This move-to-the-nearest-point operation is also referred to this as "conforming", "projecting" or "displacement"(References 15-17).

The snapping process can result in distorted elements, so the curve snapping process is repeated typically three times with a computational space smoothing inserted between each snapping step. Finally the surface snapping process is repeated typically 9 times, with a computational space smoothing between each snapping. All surface faces are quadrilaterals as can be noted from Figure 3., since all grid elements are hexahedra. A grid for a complex turbine blade geometry shown in Figure 8. demonatrates that the many corners, edges, and thin surfaces are faithfully captured even with a relatively coarse grid compared to the features. This case required under 20 minutes of elapsed time on a SUN Sparc 10 workstation and involved an 85x67x88 base grid containing 501,160 points and an input surface specified by 7,904 triangles. The final mesh contains 29,115 hexahedra and 14,364 surface quadrilaterals. Some of these elements must be broken up further for

use with current finite element structural analysis solvers as discussed later.

The above process was chosen to minimize the need for flow solver modifications. The only flow solver modifications needed for our Euler flow solver were to add a fluid/solid cell flagging capability and some logic to recognize the associated solid face boundaries. We retained our extensively calibrated strong conservation finite volume block-structured capabilities as is. Our block structured Navier-Stokes flow solver needed no modifications to use these grids to represent Euler flow (blockage) boundaries, as will be illustrated later.

Other choices could have been made at this stage to go from a stairstep grid to a grid conforming to the goal surface. Some of these choices are illustrated in Figure 9. One choice would have been to cut the stairstep cells with the goal surface, producing grid cells with other than eight hexahedral faces, as in Reference 18. This is a viable option for a solver that can handle tetrahedra or general polyhedra, and is only a minor change to the previous grid generation procedure.

Another choice would have been to cut the stairstep cells with the goal surface and feed the required areas and volumes directly to the flow solver as in References 19-20. In this technique the cells which intersect the surface (shown as dashed lines in Figure 9) are flagged to receive special treatment by the flow solver. Since most cells do not intersect the surface, this was shown in Reference 20 to incur only a minor increase in flow solver computing time. Since extensive Boolean geometric operations (clip, join, cut, union, diff) are already supported, this step would be easy to implement in the current grid generator but would require flow solver changes that are more extensive.

Yet another choice would have been to specify a stairstep design rule similar to the one discussed previously, but which defines all cells containing any solid point or intersecting any solid surface as solid. We could then "pave" or triangulate the stairstep to fill in the region out to the goal surface as in Reference 21. This introduces an unstructured grid layer at the surface and would have required a significant change in our production Euler flow solver.

Finally, we could use a combination of all the above. The extra freedom brought about could improve the grid quality near the surface and lead to "provable quality" grids as it has done for the tetrahedral mesh approach described in References 22-24 and used in the ICEM "TET mesher". Again, this would require substantial changes to our current production flow solvers.

All of these approaches start with a base grid (Cartesian, curvilinear, octree embedded, ...), determine containment of each cell, then perform operations only on cells near the surface (not throughout the volume). We call these methods **"Geometric Grid Generation"** since they all involve extensive CAD-like geometric operations which can be made very fast. Chimera grids also fit this category.

The goal surface creation and snapping operations nominally require the intersection of up to 100,000 input surface triangles with up to 100,000 base grid surface triangles, and the snapping of up to 100,000 points to the closest point on a surface of up to 100,000 triangles. The operation count for these processes can exceed 20 billion operations without using suitable data structures and search algorithms. In practice, these computations can be accomplished in a few minutes on a workstation by using suitably efficient methods (References 10-13). The snapping process is also accelerated by sorting the points to be snapped by color, and snapping them to the associated triangles sorted by color, where the color denotes to the surface to which the point is to be snapped. Randomly reordering the triangles within each color accelerated the snapping further; as soon as a random triangle gives a close distance, most remaining triangles are eliminated with a fast range check. These procedures accelerated the calculations by a factor of 6 to 12.

## QUALITY IMPROVEMENT

The grid quality at this stage is excellent everywhere except for the distorted cells near the surface, since the remaining cells are either cubes or base grid cells generated by a conventional grid generator. There often exists considerable distortion of grid cells near the surface, leading some to suspect that this process will never work in practice. This section discusses the steps taken to ensure the grid quality control which makes the process work for Euler flows.

There are two steps to improving the quality. The first is establishing the proper predictive grid measures for flow solver accuracy and stability. The second is smoothing the interior grid to enhance these measures. The measures of grid quality are based first on the particular flow solver requirements and second on some general geometric measures.

Our Euler flow solver requires a positive volume for **every cell** to run successfully. A predictive grid measure called "flatness" is presented in Appendix A for finite volume Euler flow solvers. The flatness of a cell is the height of a rectangular parallelepiped with a square base measuring one unit in width and depth and which has the same ratio of "cell volume divided by the cell face area raised to the 1.5 power" as the cell. Our Navier-Stokes flow solver, due to existing special handling of cells near solid surfaces, prefers to run with the grid as it exists at this stage with no or minimal interior smoothing. All it requires is a positive Jacobian for all cells not at the surface.

Other measures of grid quality are well known in the structures community, while less common in the fluid dynamics community. The basic measures supported here include the fundamental quad face measures (skewness, taper, warpage, convexity, and aspect ratio) and the hexahedral cell measures (edge angle, aspect ratio, twist angle) described in References 25-26 and currently used for finite element diagnostics in the PATRAN 3.0 structural analysis system.

The interior (volume) points which are within a specified integer linking amount (typically 2 or 3) of the snapped surface are also be smoothed using the technique described in Reference 7. This Laplace smoothing in physical space can be used because the block structured base grid structure is retained in the geometric grid. This physical space smoothing is less likely to fold, since it is based on the maximum principle, and also works better with high aspect ratio grid elements. The smoothing is coded in a manner which makes the block internal boundaries transparent to the smoothing process; the internal block interface boundaries "float" with the smoothing. The data synchronization process used in this smoothing is described in Appendix B. This smoothing may involve points which are on inter-block boundaries where we need to know which of the six linked points surrounding the point to be smoothed are opposite each other in the computational IJK space. A technique was developed to determine opposite points, even in arbitrary interface configurations, as required by the physical space smoothing formula. The "design rule" to determine which of the six linked points are opposite is based on the simple observation that opposite points in different blocks do not share a common face, while all other pairings of the six linked points share a common face. The "same-face" logic requires only one page of coding. Any points which are linked to more than six other points, such as at the centerline of a cylindrical grid, are not moved

In some situations, portions of the base grid surface which are outside the added input surface must be smoothed as well, while constraining them to lie on the base grid surface. Without this extra smoothing, grid folding may occur on original surfaces at the intersection of the snapped surface due to the differing requirements of the stair-step containment decision (which is a 3D criterion involving 8 points and 12 edge segments) and the original surface points taken individually. This behavior requires that we smooth the original surface near the snapped surface. This smoothing must smooth

the original surface points distribution while keeping them within the original surface. It must also keep the original surface points which lie on any original surface sharp edges on those sharp edges. It must also not move original surface corner points. Those original surface points within a specified integer linking amount (typically 2 or 3) of the intersection of the snapped surface with the original surface are so smoothed. For the surface smoothing, the smoothed point is moved to the average position of it's linked points and then it is snapped to a saved copy of the original surface such that the outward normals of the snapped and original surfaces lie within 90 degrees. This normal logic prevents "snapthrough" which may occur when the smoothing the base surface grid near the trailing edge of a thin nacelle cowl where the circumferential grid size is much larger than the cowl thickness. In this case the simple nearest surface logic moves the point through the thin trailing edge region to the wrong surface .Edge points are smoothed by moving them to the average of the two linked edge points, and then snapped to a copy of the original edges. This computational space smoothing works well on the original surface and edges. There is no need to separately treat each edge path and each surface as in the snapping module. All that is necessary is to separate the corner, edge, and surface points and snap to either the set of all edge paths or the set of all surfaces.An example of this extra smoothing can be noted on the top rear of the core cowl adjacent to the pylon in Figure 5.

The centroid/volume weighted Laplace smoothing described in Reference 27 was also tried in the hope that the closer the smoothing matches the flow solver, the better. The flow solvers used so far have all been finite volume flow solvers. This approach did not prove immune to occasionally producing cells with negative volumes. All of the above smoothing methods can produce negative volumes, particularly near concave surfaces. In that case, all of the support points and/or volume centriods can lie outside the flow region and any weighting (be it computational space weighting, physical space weighting, or centroid/volume weighting) can move a base point outside the fluid region.

The major operation counts for the smoothing step consist of up to 1,000,000 Laplace volume smoothing operations repeated typically 10 times and up to 100,000 surface snapping operations repeated typically 10 times. These operations can all be completed in a few minutes on a workstation.

One characteristic of the current geometric grid generation method is that the vast majority of grid cells are identical to the base grid cells, which are presumably of good quality. Only a few cells near the surface can have difficulties as described above. One solution to this is to treat only these relatively few cells individually along the lines described in Reference 15. In the pending implementation of this method, the cell having the lowest flatness value is chosen and improved by solving a optimization problem where the function to be maximized is the minimum flatness of the cell and all directly abutting cells. The variables are the interior node positions of the cell to be improved.

Even the smoothed distorted elements near the surface currently often exceed the bounds set by finite element structural solvers. Structural solvers may require a positive Jacobian at as many as 27 integration points. The geometric grid generation process can be modified by breaking up each significantly distorted element into a collection of less distorted (more convex) elements using published methods. One method, implemented here, breaks up distorted hexahedra into a collection of wedges, pyramids, and tetrahedra. The applicability of this method depends on whether or not the structural solver being used supports pyramids; some do not (at least easily). Pyramids are a crucial matching element between tetrahedra and hexahedra. Sometimes this breakup approach leads to the propagation of "pencils" of wedges all the way from a surface to an exit elsewhere on the surface. Another method in Reference 17 breaks up distorted hexahedra into a limited number of hexahedra all lying near the surface. The idea is to break up the distorted elements without adding a large number of new elements.

Flow solver data structures. - The NASTAR Navier-Stokes flow solver uses point "flag" values to control boundary conditions. It examines a group of point flags to deduce where solid cells are. The minimum solid cell configuration is a 2x2x2 cell; there must be at least two solid cells in each of the three coordinate (index) directions for a solid to be detected. The Ni Euler flow solver uses cell "flag" values to indicate solid cells, and uses cell information for the two cells straddling a face to determine the flow algorithm (flow or no flow) to use for the face. It also uses an "index table" to assign boundary conditions to each block face. This requires that all inflow and outflow cell faces lie on a block boundary, but allows solid regions as small as one cell thickness in each direction. Both of these techniques require the storage of flags approximately equal in number to the number of grid points. An alternate technique is to assign flags to face elements of the cells. Internal faces which share a common set of four points need only be represented once, so this scheme requires the storage of flags approximately equal to three times the number of grid points. However, this allows a thin solid (such as a jet engine mixer) to be represented as a membrane of zero thickness. It also allows the treatment of non-manifiold geometries and helps to keep track of inflow and outflow boundary conditions. The geometric grid generator can produce grids suitable for all three of these flow solver data structures.

## DEMONSTRATION & VALIDATION

The first validation is the Euler transonic flow calculation in a 2-dimensional channel containing a circular bump known as the "Ni bump" case. The results from a conventional curvilinear grid and from the geometrically generated grid are presented in Figure 10. The geometric grid is distorted at the bump surface, yet it accurately predicts the transonic flow even on this relatively coarse grid at a near-choked condition.

The second validation is the Euler flow over a wing/pylon/nacelle where the pylon was geometrically added to an existing wing/nacelle grid as illustrated in Figure 11. The excellent agreement between the Euler calculations and NASA experimental measurements from Reference 28 at a transonic section of the nacelle very near the pylon/nacelle intersection is shown in Figure 12. Another case (not shown here) at a lower free stream Mach number shows excellent agreement with the VSAERO subsonic panel method solution from Reference 29 and NASA experimental data for this case where substantial wing/pylon effects exist.The author is grateful to Ron-Ho Ni who modified his Euler flow solver described in Reference 30 to accommodate the flagging of fluid and solid cells for these calculations, and to Bill Siddons, Bob Miller, and Ed Migliaro who provided the wing/nacelle base grid and the pylon geometry definition.

This technique was also used to predict the inviscid flow from the inlet of a helicopter, into a plenum, into a bellmouth, and finally into the inlet of a gas turbine engine. A fine grid is shown in Figure 13, and flow direction vectors from an earlier coarse grid (without a cylindrical pipe through the plenum which was added to the fine grid) are shown in Figure 14.

As a demonstration of use with a Navier-Stokes flow solver, the grid generator was applied to predict the flow blockage produced by inserting a probe in a compressor cascade blade row. The probe, defined by an IGES file, was read into the Unigraphics CAD program which produced a sterolithography file specifying the input surface. This probe was added in an Euler flow mode (slip flow boundary condition on the probe) to a viscous cascade base grid. The base grid surface and the snapped grid representation of the probe (as a shaded image) are illustrated in Figure 15. The author is indebted to Tom Rogers, Bob Zacharias, and Chae Rhie for providing the probe surface definition, the compressor passage base grid, and instruction in running the NASTAR Navier-Stokes flow solver described in Reference 31.



ORIGINAL PAGE IS OF POOR QUALITY

The final demonstration addresses structural grid generation. The example shown in Figure 16 is a MARC finite element solution for the VonMises stress in a loaded bracket assembly. The distorted elements have been broken up mainly into wedges. This example is included to demonstrate that structural grids can be successfully generated by this method, so that coupled fluid-structural solutions will be possible using geometric grids.

## FUTURE DIRECTIONS

When this work was started, it was not known how good the flow solver predictions would be with these grids. The accuracy has exceeded our initial expectations and the extensive geometric operations required were successfully accelerated to reasonable times on a workstation.

It is anticipated that this technology can be extended to generate viscous grids by the process sketched in Figure 17 with less technical risk than was incurred in generating the Euler grids. The steps include generating a hyperbolic marching grid to a finite distance from the body as in References 32-34, and then constructing a geometric Euler grid from the outer hyperbolic surface. Then the geometric grid nodes on this surface are interpolated back to the body using the hyperbolic grid cells as an interpolation basis, finally the combined grid is smoothed. Many capabilities needed for this viscous extension are already coded for current grid generation capabilities. Examples include modules for computing surface normals and modules to determine the distance from a point to the nearest surface (including the surface identification number and the matching outward normal requirement). Combining these with the considerable literature and existing programs for hyperbolic marching grid generation is looked forward to with anticipation. It is noted that these viscous grids will contain extra directionally structured hexahedral collections of cells and will require either that our flow solvers be modified to match or that other flow solvers be used.

To reduce the current dependence on using simple conventionally generated base grids, it is necessary to extend the geometric grid generation process to include adaptive grid technology. It is expected that this will be straightforward due to the cell-by-cell approach used in the current implementation. Then the initial construction of a base grid may be replaced by a simple automatic coarse cubical grid enclosing the flow region, to be enhanced locally as needed.

It is noted that crucial enabling technologies allowed this work to be accomplished. These enabling technologies include fast workstations with large core and disk storage, and the attendant utilities such as make and dbxtool to manage and debug large coding projects. Another significant enabler is the ability to view complex surface images on the desktop, such as with the SUN ZX card and "leotool" graphics program. This project would have been unthinkable with the limited core shared-mainframe hardware, next-day graphics, and remote character terminals of ten years ago!

## CONCLUSIONS

This paper demonstrates that highly automated geometric hexahedral grid generation for unique complex geometries can be accomplished in **one hour** on a workstation. The resulting grids yield demonstrated accurate Euler flow simulations as compared to experimental data, panel method calculations, and Euler flow simulations with standard curvilinear grids.

"**Geometrical Grid Generation**" is suggested as a descriptor for approaches that determine containment on a base grid and then operate only on cells near the surface to complete the gridding.

## REFERENCES


1. Boerstoel, J.W., Spekreijse, S.P., and Vitagliano, P.L., "The Design of a System of Codes for Industrial Calculations of Flows around Aircraft and Other Complex Aircraft Configurations," 10th AIAA Applied Aerodynamics Conference, Paper # AIAA-92-2619-CP, Palo Alto, CA, June 1992.
2. Rubbert, P., "CFD and the Changing World of Airplane Design," AIAA Wright Brothers Lecture, Anaheim CA, Sept 1994.
3. Blake, M. W., Kerr, P. A. and Thorp, S. A., "NASA Geometry Data Exchange Specification for Computational Fluid Dynamics (NASA IGES)," NASA-RP-1338, April 1994.
4. Remotique, M. G. and the NGP Team, "The National Grid Project: Making Dreams into Reality," 4th International Conference on Numerical Grid Generation in Computational Fluid Dynamics and Related Fields, pp. 429-439, Pineridge Press,Swansea, United Kingdom, April 1994.
5. Maskew, B., "Prediction of Subsonic Aerodynamic Characteristics: A Case for Low-Order Panel methods," Journal of Aircraft, Vol. 19, No. 2, pp.157-163, 1982.
6. Akdag, V., and Wulf, A., "Integrated geometry and Grid Generation System for Complex Configurations," Software Systems for Surface Modeling and Grid Generation, NASA CP-3143, pp. 161-171, Hampton, VA, April 1992.
7. Zacharias, R. M., Ives, D. C., and Siddons, W. D. Jr., "3D Grid Generation Tools," AIAA/SAE/ASME/ASEE 26th Joint Propulsion Conference, Orlando, Florida, July 1990.
8. Glassner, A. S., "An Introduction to Ray Tracing," Academic Press, Harcourt Brace Jovanovich, New York, New York, 1989.
9. LaBozzetta, W,F, et al, "MACGS - Towards the Complete Grid Generation System," AIAA-94-1923-CP.
10 Franklin, W. R., Chandresekhar, N., Kankanhalli, M., Akman, V., and Wu, P., "Efficient geometric Algorithms for CAD," Geometric Modeling for Product Engineering, pp. 485-498, Eslevier Science Publishers BV, North-Holland.
11. Samet, H., "The Design and Analysis of Spatial Data Structures," Addison-Wesley, Reading, Massachusetts, 1989.
12. Bonet, J. and Peraire, J., "An Alternating Digital Tree (ADT) Algorithm for 3D Geometric Searching and Intersection Problems," International Journal for Numerical Methods in Engineering, Vol. 31, pp. 1-17, 1991.
13. Sedgewick, R., "Algorithms," Addison-Wesley Publishing Company, Reading MA, 1983.
14. Hamann, B., "A Data Reduction Scheme for Triangulated Surfaces," Computer Aided Geometric Design, Vol. 11, pp.197-214, 1994.
15. Taghavi, R., "Fully Automatic Grid generation from CAD on Vector-Parallel and Massively Parallel Supercomputers," Societe des Ingenieurs de l'Automobile, SIA Conference, Massay-Palaiseau, France, February 1993.
16. Burantynski, E. K., "A Fully Automatic Three-Dimensional Mesh Generator for Complex Geometries," International Journal for Numerical Methods in Engineering, Volume 30, pp.931-952, 1990.
17. Schneiders, R., and Oberschelp, W., "Automatic Generation of Hexahedral Element Meshes for the Simulation of Metal Forming Processes," pp. 223-233, in "Numerical Grid Generation in Computational Fluid Dynamics and Related Fields," Wetherill, N. P., et al., Swansea, Wales, April 1994.
18. Wang, Z. J., "A Fully Conservative Structured/Unstructured Chimera Grid Scheme," AIAA 33rd Aerospace Sciences Meeting & Exhibit, Paper# AIAA-95-0671, Reno NV, Jan 1995.
19. Landsberg, A.M., Young, T,R.,Jr., Boris, J., "An Efficient, Parallel Method foe Solving Flows in Complex Three-Dimensional Geometries.", AIAA 32nd Aerospace Sciences Meeting & Exhibit, Paper # AIAA-94-0413, Reno, NV, January, 1994.
20. Melton, J. E., Berger, M. J., Aftosmis, M. J., and Wong, M. D., "3D Applications of a Cartesian Grid

Euler Method," AIAA 33rd Aerospace Sciences Meeting & Exhibit, Paper# AIAA-95-0853, Reno NV, Jan 1995.
21. Kao, K. H., and Liou, M. S., "Direct Replacement of Arbitrary Grid-Overlapping by Nonstructured Grid," AIAA 33rd Aerospace Sciences Meeting & Exhibit, Paper# AIAA-95-0346, Reno NV, Jan 1995.
22. Moore, D., "Simplicial Mesh Generation with Applications," Department of Computer Science, Cornell University, Ithaca, New York, December 1992.
23. Moore, D. and Warren, J., "Multidimensional Adaptive Mesh Generation," Rice COMP TR90-106, Department of Computer Science, Rice University, Houston, Texas, February 1990.
24. Moore, D. and Warren, J., "Adaptive mesh generation II: Packing Solids," Rice COMP TR90-139, Department of Computer Science, Rice University, Houston, Texas, March 1991.
25. Robinson, J. and Haggermacher, G., "Element Warning Diagnostics," Finite Element News, Volume 3, pp.30-33, June 1982; Volume 4, pp.19-23, August 1982.
26. Robinson, J., "Some new Distortion Measures for Quadrilaterals," Finite Elements in Analysis and Design, Volume 3, pp. 183-197, 1987.
27. Stephenson, M. B., Hsu, S. S., and Benzlet, S. E., "Automatic Mesh Generation for Free-Form Deformation Solids," COMPUTERS IN ENGINEERING 1990, Proceedings of the 1990 ASME International Computers in Engineering Conference and Exposition, Boston, Massachusetts, August 1990, pp. 277-286.
28. Carlson, J.R. and Compton, W.B., "An Experimental Investigation of Nacelle-Pylon Installation on an Unswept Wing at Subsonic and Transonic Speeds," NASA Technical Paper 2246, February 1984.
29. Lord, W. K. and Zysman, S. H., "VSAERO Analysis of a Wing/Pylon/Nacelle Configuration," AIAA/ASME/SAE/ASEE 22nd Joint Propulsion Conference, Paper# AIAA-86-1523, Huntsville, AL, June 1986.
30. Ni, R. H., "A Multiple-Grid Scheme for Solving the Euler Equations," AIAA Paper 81-1025, Palo Alto, California, June 1981 and AIAA Journal, Volume 20, No. 11, November 1982.
31. Rhie, C. M., "Pressure Based Navier-Stokes Solver Using the Multigrid Method," AIAA Journal, Volume 27, Number 8, pp. 1017-1018, August 1989.
32. Karman, S. L. Jr., "SPLITFLOW: A 3D Unstructured Cartesian/Prismatic Grid CFD Code for Complex Geometries," AIAA 33rd Aerospace Sciences Meeting & Exhibit, Paper# AIAA-95-0343, Reno NV, Jan 1995.
33. Kallinderis, Y. and Ward, S., "Prismatic Grid Generation for 3-D Complex Geometries," AIAA Journal, Vol. 31, No. 10, pp.1850-1856, Oct 1993.
34. Chan, W. M. and Buning, P. G., "A Hyperbolic Surface Grid Generation Scheme and its Applications," AIAA Paper 94-2208, AIAA 25th Fluid Dynamics Conference, Colorado Springs, Colorado, 1994.
35. Press, W. H., Flannery, B. P., Teukolsky, S. A., Vettering, W.T., "Numerical Recipes", Cambridge University Press, New York, New York, page 146, 1986.
36. McConnell, S., "Code Complete," Microsoft press, Redmond WA, 1933.
37. Monoit, R, "**ftnchek**," available by anonymous ftp from research.att.com (192.20.225.2) in the directory /netlib/fortran or by electronic mail by sending the message "send ftnchek from fortran" to the internet address: netlib@ornl.gov

# APPENDIX A - FLATNESS

This appendix defines a "flatness" measure to **predict** the suitability of a grid cell for use with finite volume solvers. Finite volume solvers distribute the flux entering all the faces to the volume contained within the cell, so a likely measure relates the volume to the total face area. Since volume has three dimensions, and area has two dimensions, we choose a form of relation involving volume/area$^{1.5}$ to produce a dimensionless measure. We also desire the measure to vary from -1 to +1 and to have a simple physical interpretation. Such a definition of cell "flatness" is:

$$VA=volume/(|area|/6)^{1.5}=flatness/((1+2*|flatness|)/3)^{1.5}$$

This flatness is physically interpreted as the height of a rectangular cube with a unit square base which has the same value of [volume/|area|$^{1.5}$] as the element. The rectangular cube has volume=flatness and area=(2+4|flatness|). The 6 in the formula ensures a flatness range of -1 to +1. It is our experience that as long as the minimum flatness exceeds .02, the Ni Euler flow solver of Reference 30 does not encounter grid difficulties. A flatness of -1 represents a cube whose faces are oriented counter-clockwise-in (negative volume). Face areas are positive by definition. A ratio of +1 corresponds to a cube whose faces are oriented counter-clockwise-out (positive volume). A flatness of zero corresponds to a hexahedron with no volume, for example a hexahedron with all its' vertices lying within a plane. The rate of change of VA with respect to flatness is zero at |flatness|=1, so the simple VA ratio does not provide good resolution for cells with |flatness|~1. The flatness equation can require the solution of a cubic equation. For |VA|<.707, we use Newton-Raphson iteration to solve for VA, otherwise we use the method described in Reference 35. The two-way implementation of the relation between VA and flatness is given below to encourage wider use of the flatness measure. This FORTRAN code is rapid, stable, and accurate to about +/- .000001.

```
      FUNCTION FLATNESS (VA)
      REAL*4     FLATNESS,VA,A1,A2,A3,Q,R,TH,S,ERR
      INTEGER*4 I
      IF(ABS(VA) .LE. .707) THEN
         FLATNESS=0.
         DO 1 I=1,8
         ERR=FLATNESS/(SQRT((1.+2.*ABS(FLATNESS))/3.))**3-VA
         IF(ABS(ERR).LT.1.E-7) GOTO 2
         S=SQRT((1.+2.*ABS(FLATNESS))/3.)
1        FLATNESS=FLATNESS-ERR*3./(1.-ABS(FLATNESS))*S**5
      ELSE
         A1=(1.500-3.375/VA**2)
         A2=.750
         A3=.125
         Q=SQRT((A1**2-3.*A2)/9.)
         R=(2.*A1**3-9.*A1*A2+27.*A3)/54.
         TH=(ACOS(R/Q**3))
         FLATNESS=-2.*Q*COS(TH/3.+3.14159265*4./3.)-A1/3.
      ENDIF
2     FLATNESS=SIGN(FLATNESS,VA)
      RETURN
      END

      FUNCTION VA (FLATNESS)
      REAL*4    VA,FLATNESS
      VA=FLATNESS/(SQRT((1.+2.*ABS(FLATNESS))/3.))**3
      RETURN
      END
```

# APPENDIX B - GENERIC ISSUES

This appendix contains discussions of technical issues which involve all of the above processes.

## Tolerances

In the geometric grid generation process, we first determine which points are coincident in the base grid, and keep track of them as a single unit. After this determination all canceling of cubes or quads takes place using only pointers, and not the coordinates themselves, to guarantee insensitivity to tolerances. We thus separate the geometry from the topology. This is a key issue for single precision manipulation of inviscid grids where the length scales can vary by five orders of magnitude, such as for a wing/pylon/nacelle/fuselage. Essentially, the coincidence of base grid points is checked initially with a zero tolerance and then all following operations on the base grid are performed with zero tolerance settings. This presumes that the input base grid is constructed that any coincident points are coincident, not just near each other. The ICEM grid generator of Reference 6, used here for many of our base grids, can produce non-coincident points, but it also produces an integer face connectivity table which can be used to enforce strict coincidence of face points. This then allows us to operate with zero tolerances and create single precision Euler quality grids for a fuselage/wing/nacelle/pylon installation flow simulation involving grid element sizes which vary by 5 orders of magnitude.

## Data Synchronization

In handling multi-block grids, the a point may be shared between multiple blocks at common interfaces. The problem then arises of how to keep all of these different images of the same point identical. One way, commonly used in CAD systems, is to deal exclusively with pointers to a triad of coordinates and store the coordinates only once. This is efficient for CAD, which has relatively few such points, but can be inefficient for multiblock grids with often millions of points. For multiblock grids only relatively fewer grid points lie on block interfaces than the total number of grid points, so we only must handle these fewer points in a synchronous manner. The technique used here is simple to implement, and guarantees that all images of a point match. The method allows a "read" of any of the equivalent points, and when a "write" is needed all images of the point (kept in a separate list) are written. This guarantees that all images of the point are identical. This is somewhat simpler to implement than referring to all interface points by pointers or reference and all interior points by value. Thus all points are referred to by value for read operations without any special handling and only the write operation need be modified from a code which does not enforce synchronicity. For write we intercept the standard write operation and test if the point being written is an interface point. If it is not an interface point, it is simply written as before. If it is an interface point, all the pointers to the corresponding image points are used to write the identical value at the corresponding locations. In essence, we read by value and write by image references. This method can be described as "read any / write all". In essence, we use I,J,K,L computational space indices for points interior to the blocks, and the equivalent of pointers to block face points.

The block boundary image point determination is made by examining the supplied base grid for points having the exactly coincident coordinates. Pointers to those points which share the same location are saved in an "images" list which is constructed for rapid recall of all pointers equivalent to a given pointer. In practice, some base grid generators may not produce exact grid point coincidence at common boundaries. In this case we either may either merge block boundary points which are coincident within a tolerance or merge block boundary points specified by a block face matching table produced by the base grid generator. At least one base grid generator, Reference 4, guarantees by construction coincident points at common block interfaces.

## Clipping

The surface to be clipped can be non-manifold and things still work perfectly, since only one clipped surface triangle is processed at a time. All that is required is that the clipping surface (which comes from skinning a usually well-behaved grid) be nice. The logic used always gets the edge intersections from low to high, so that two triangles sharing a common edge are treated identically. The logic also completely avoids the use of the square root operator. An approximation is used to simplify the process: namely the clipped surface edges are intersected with the clipping surface and the clipped surface edge intersections are connected. No account is taken of the variation (other than linear) of the clipping surface within the triangles to be clipped. No account is taken for the intersection of multiple parts of a surface within the triangles to be clipped. This method cannot be used for Boolean unions, as the unions will not join continuously. However, this method is often fully suitable for trimming the goal surface with the base grid skin. The code is fast, requiring only 21 seconds on a Sparc 10 for a clipping a surface of 29,564 triangles with a clipping surface defined by 29,552 triangles.

## Cache Coherence

When the geometric grid generator works with a structured base grid, the information needed by a flow solver is still as adjacent in memory as it is for a structured grid generator. Then, with modern workstations having large amounts of (sometimes even two level) cache memory, it is believed that cache misses will be small and the present grids will show a negligible increase in CPU time requirements due to cache misses and paging. For adaptive grids it is anticipated that if we place the embedded grid storage for each coarse grid cell adjacent to the data for the coarse grid cell, we will likewise have efficient cache memory usage.

## Visualization

The higher level of operations in this process, compared to current interactive CAD based approaches, requires that surface visualization be available to inspect the results. Without this visualization, it is very hard to discern where the process failed. This is particularly important when checking the input surface file and the base grid file, and when deciding which operations to perform. For example, if the input surface is not closed, it must be zippered as described above. If one forgets to check for openness (either with diagnostic code or with a quick visual look), then the automatic blocking process will fail badly. Visualization is crucial at times like this. The code currently is controlled by standard UNIX command lines with options enabled or set by command line arguments. A few command-line responses are required within the runs; these are being removed as feasible for the production version. A GUI shell can then be added later to further simplify use.

## Coding Quality

The approach described here consists of a sequence of high-level operations, and it is imperative that each operation work **perfectly** or the overall results will be useless; all it takes is one bad cell out of a million cells to cause the flow calculation to diverge! This places stringent demands on the approximately 70,000 lines of FORTRAN code which implement this process. It is necessary to code very defensively, checking data and **assumptions** whenever possible. The major operations are currently separated by files (which one module produces and the next module reads) to decouple the operations and to simplify debugging.

The main type of bugs currently encountered are "concept bugs" which typically involve implicit assumptions overlooked in the initial formulation. Often an assumption is made that one has

imagined all combinations possible, and accounted for them, but in fact some have been missed. This often happens in topological circumstances. One example is the assumption that each edge of a triangle in one surface being cut by another surface can intersect the other surface in only one point. This assumption led to code which uses containment logic to slice the triangle, and if both end points of an edge are on the same side of the cutting surface concludes that the edge does not intersect the surface. This works most of the time, but fails on occasions when the assumption is violated. The fix is to use intersection logic, rather than containment logic, in this particular case. Concept bugs are generally harder to rectify, since they affect the basic formulation.

Coding memory allocation errors are prevented by an interface between the programs and the FORTRAN *malloc* and *free* calls as suggested in Reference36. Other coding errors tend to be removed by using different compilers and the ftnchek FORTRAN source code checker from Reference 37. The maximum optimization level that is used on the SUN is opt2, since opt3 produced erratic results for pointer variables resulting from dynamic memory allocation and for embedded "DO" loops containing "GOTO" statements pointing to a common "CONTINUE" statement. With these quality conformance tools, combined with defensive coding, the primary source of errors is concept bugs.

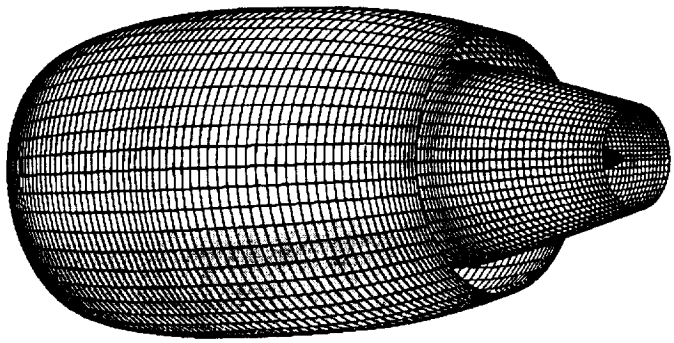


Figure 1. Base Grid Surface for a Nacelle

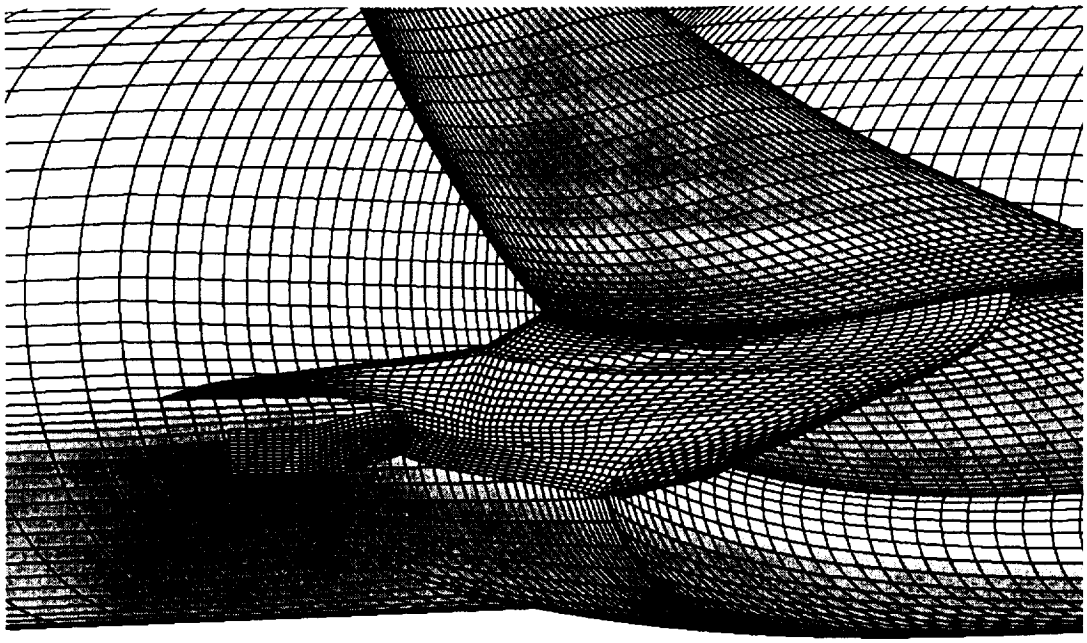


Figure 2. Fuselage/Wing/Pylon Surface from VSAERO File

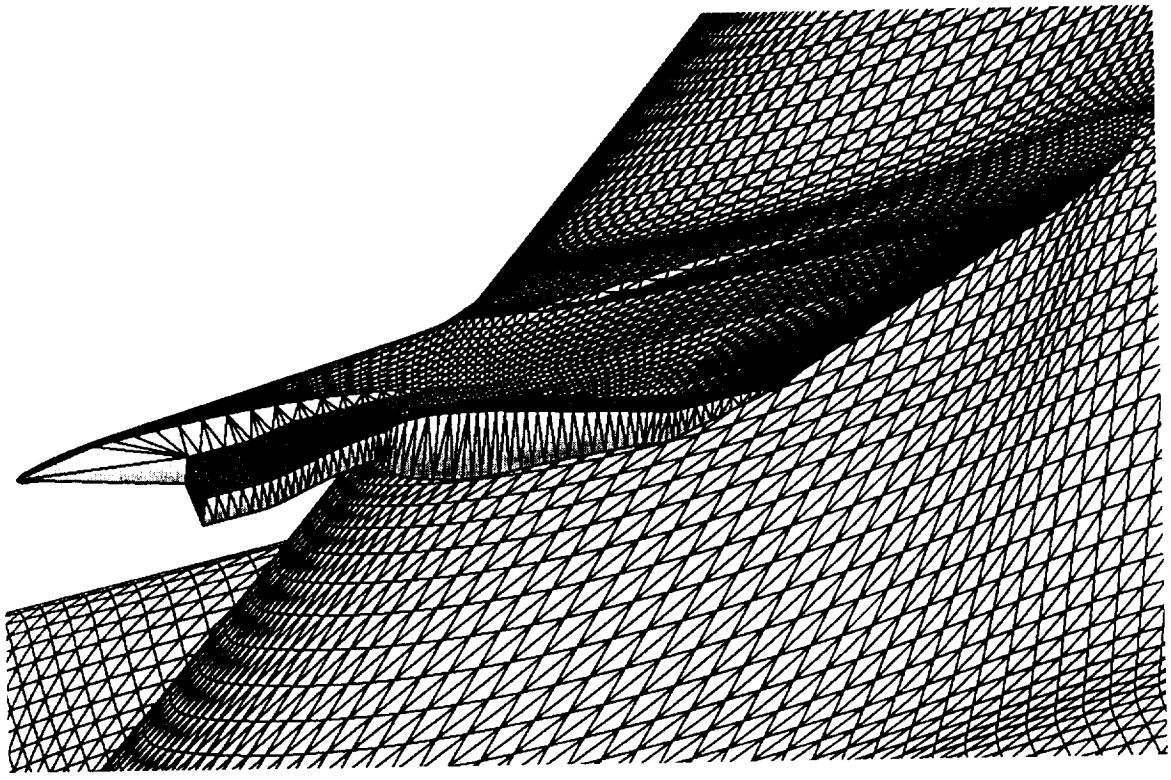


Figure 3. Closed Surface

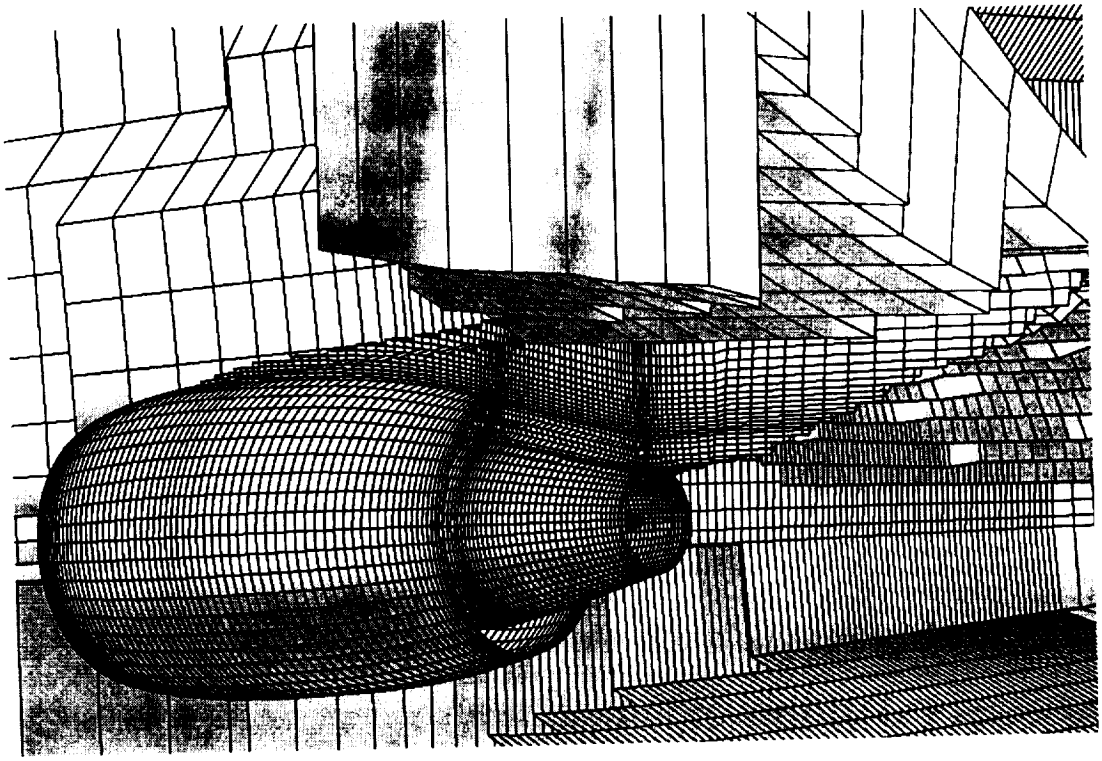


Figure 4. Stairstep Grid Surface

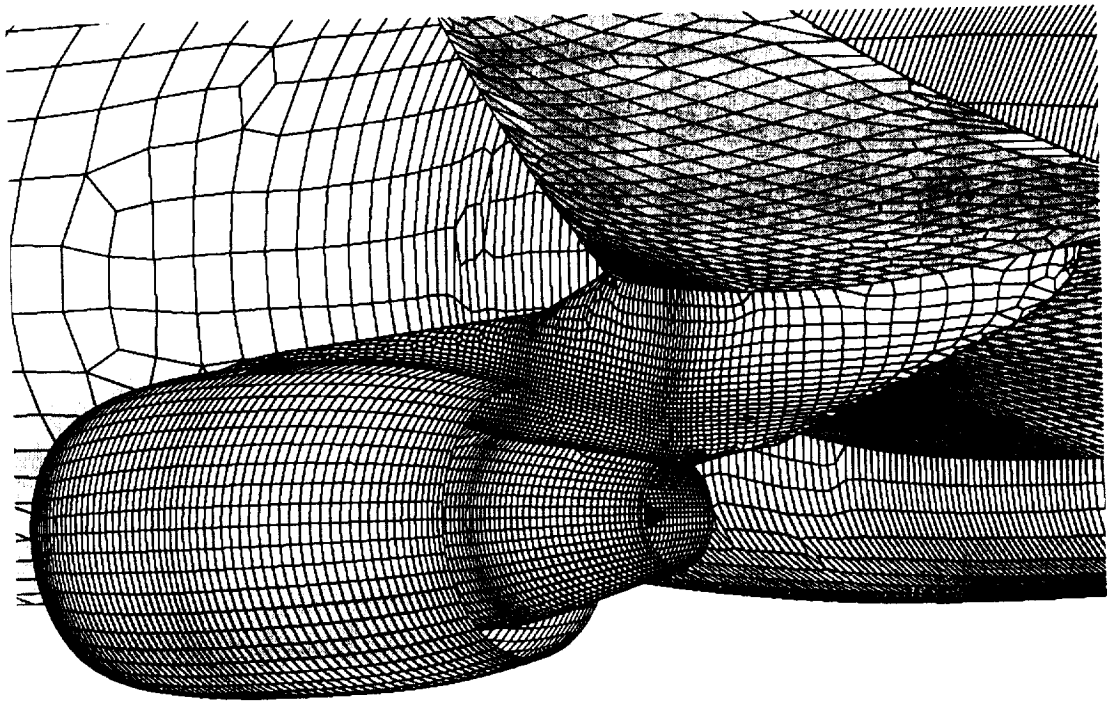


Figure 5. Snapped and Smoothed Grid Surface

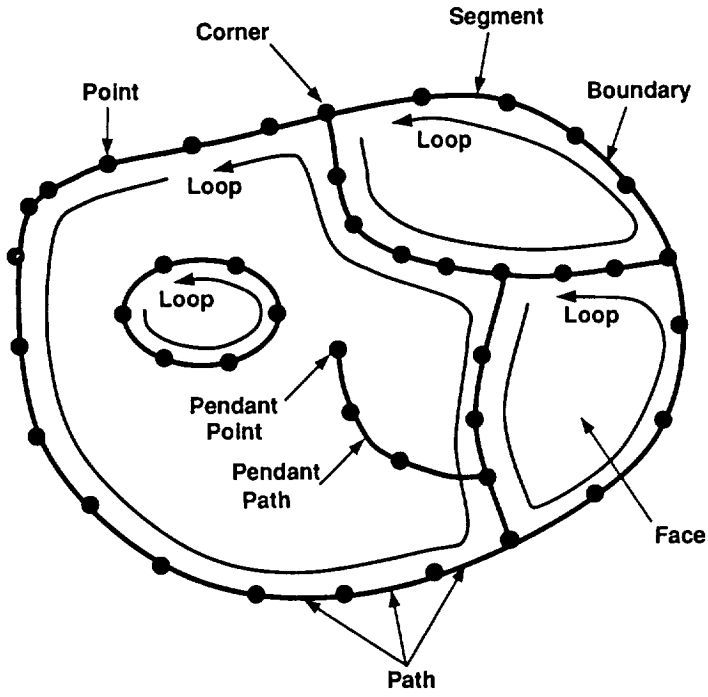



Figure 6. Grid Features

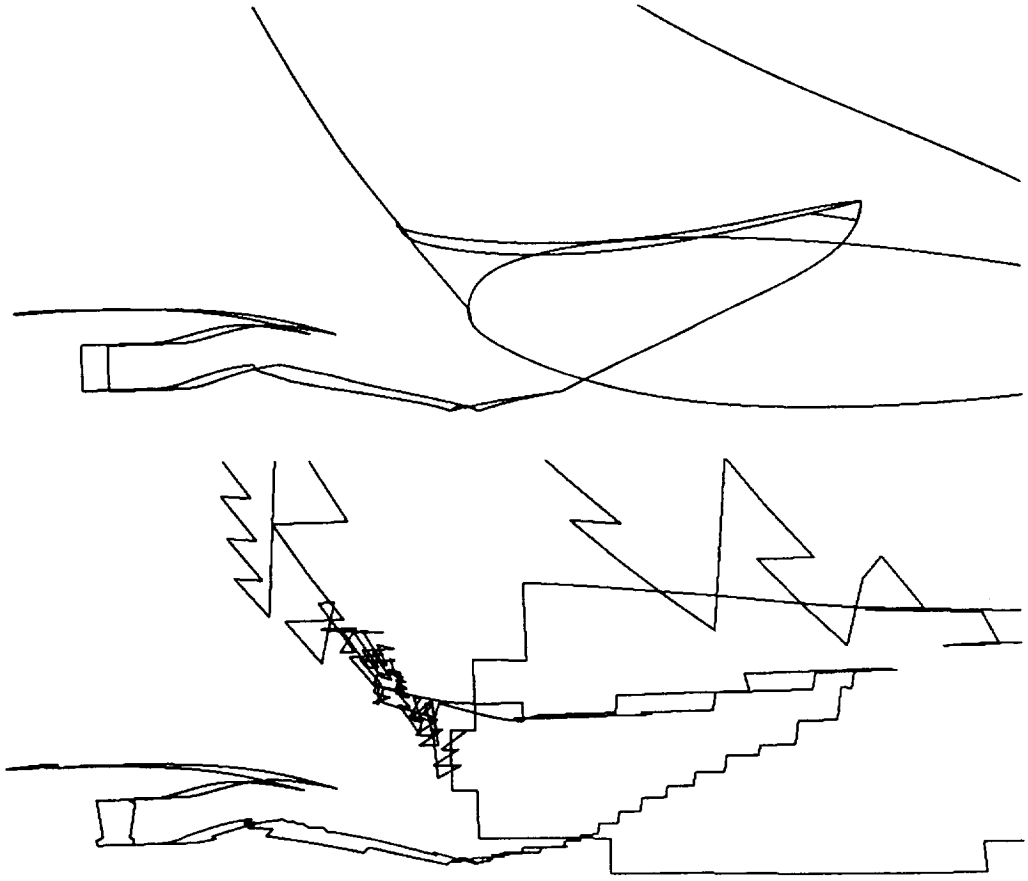


Figure 7. Associated Paths

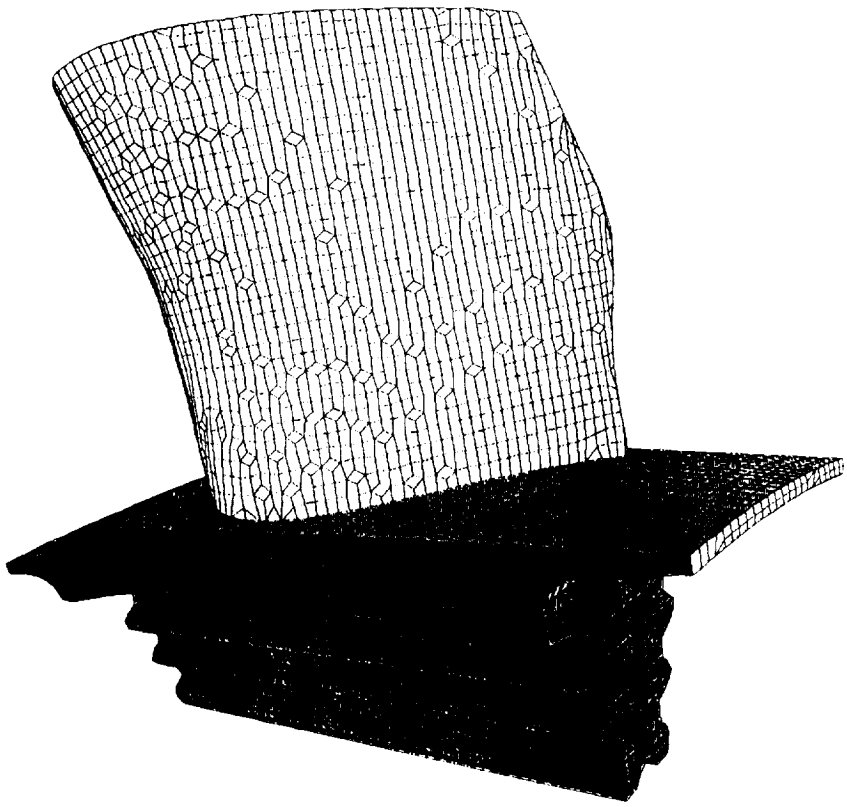


Figure 8. Turbine Blade Grid Surface

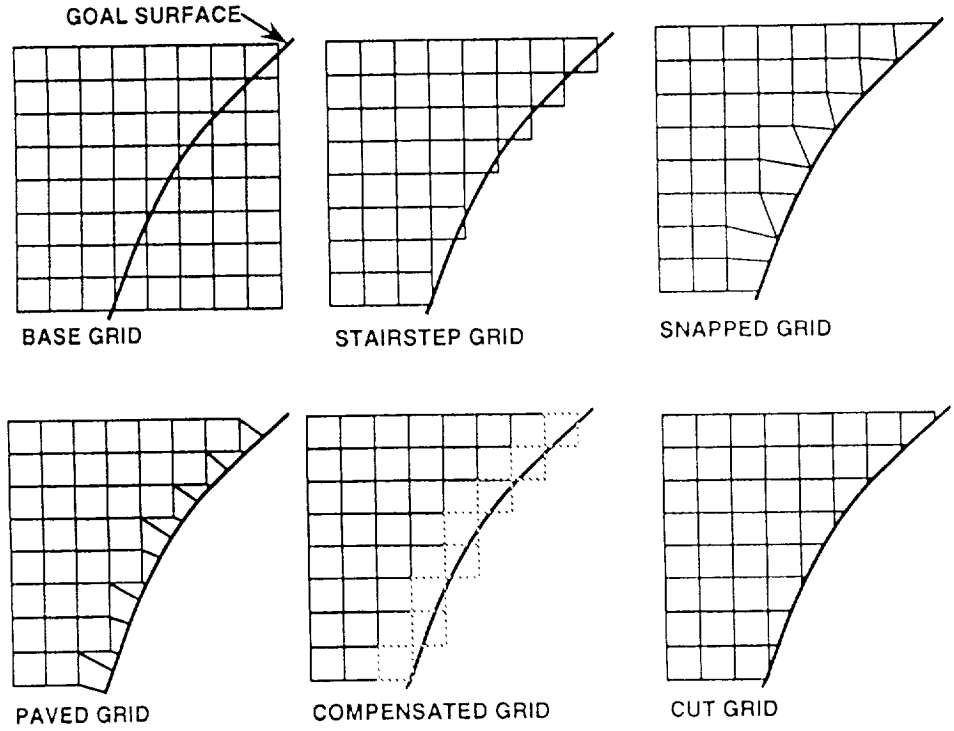



Figure 9. Surface Cell Methods

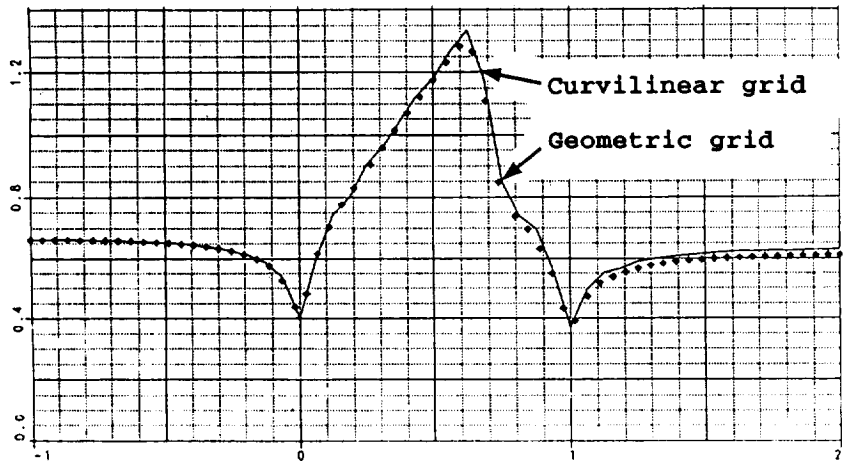



Figure 10. Flow over a Circular Bump in a Channel

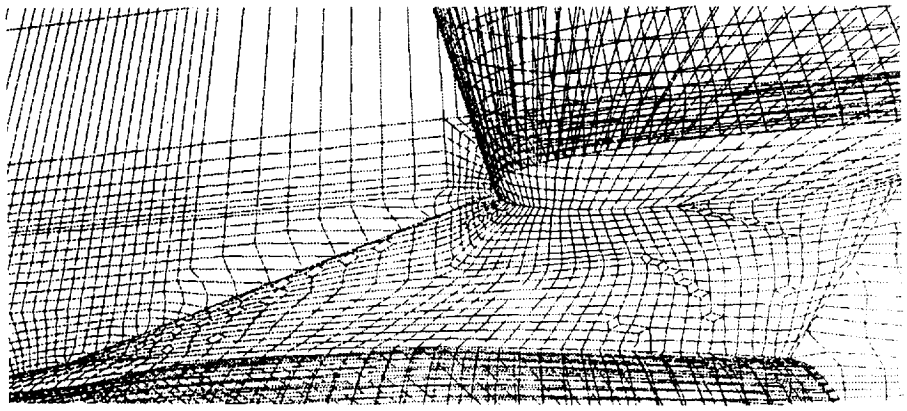


Figure 11. Pylon added to Wing/Nacelle Grid

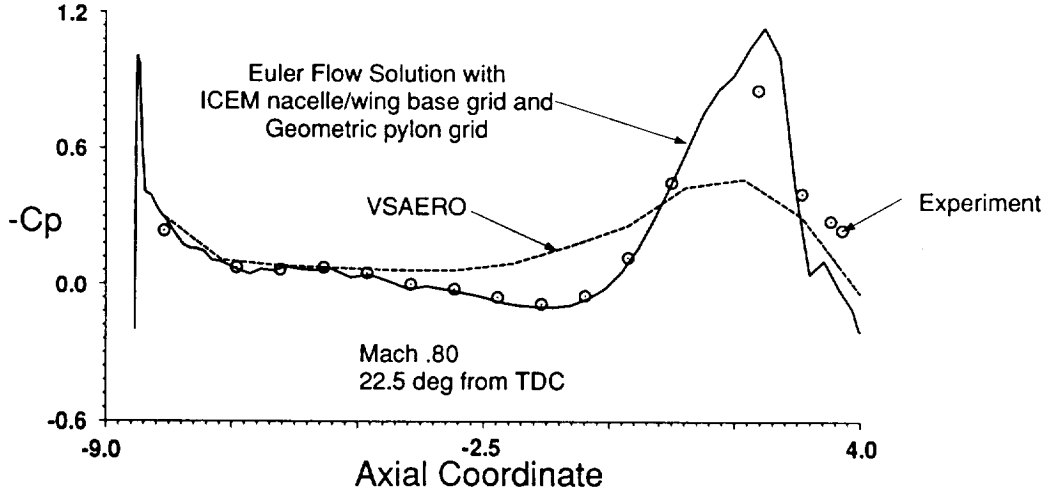



Figure 12. Wing/Pylon/Nacelle Flow Comparisons

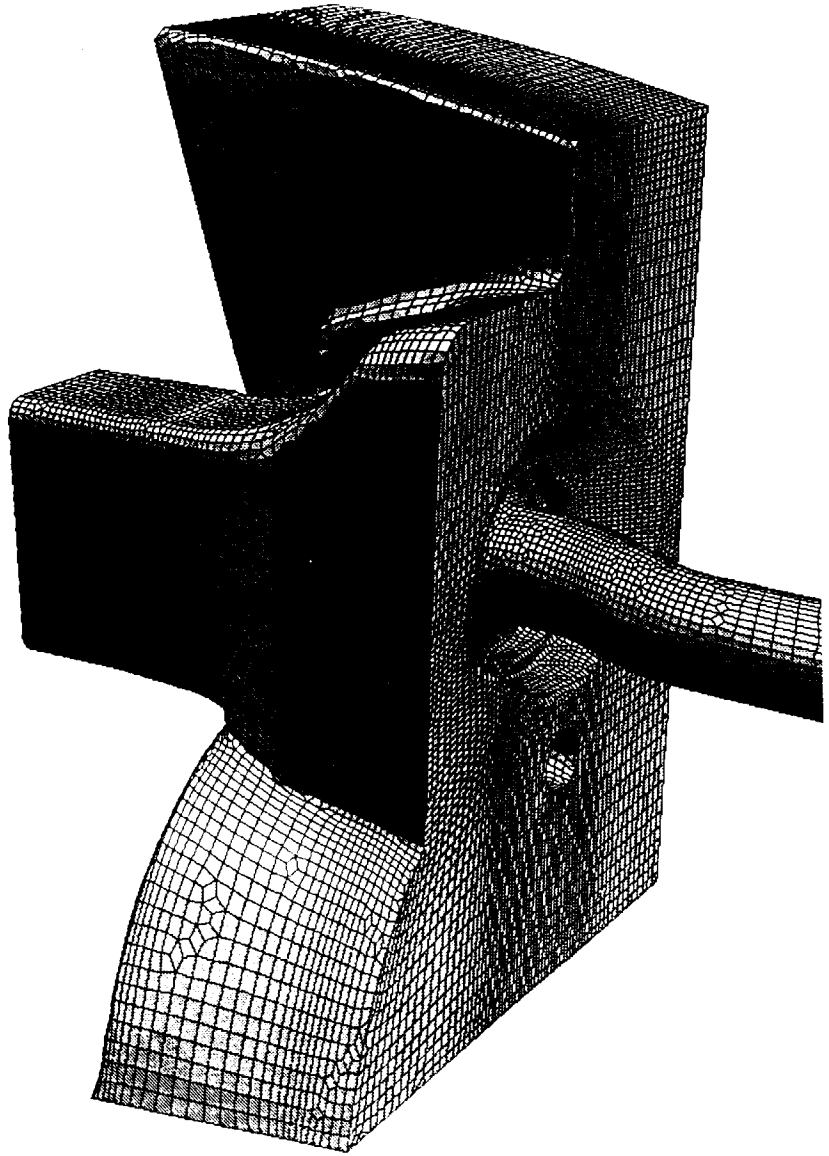


Figure 13. Helicopter Inlet Flow System Grid

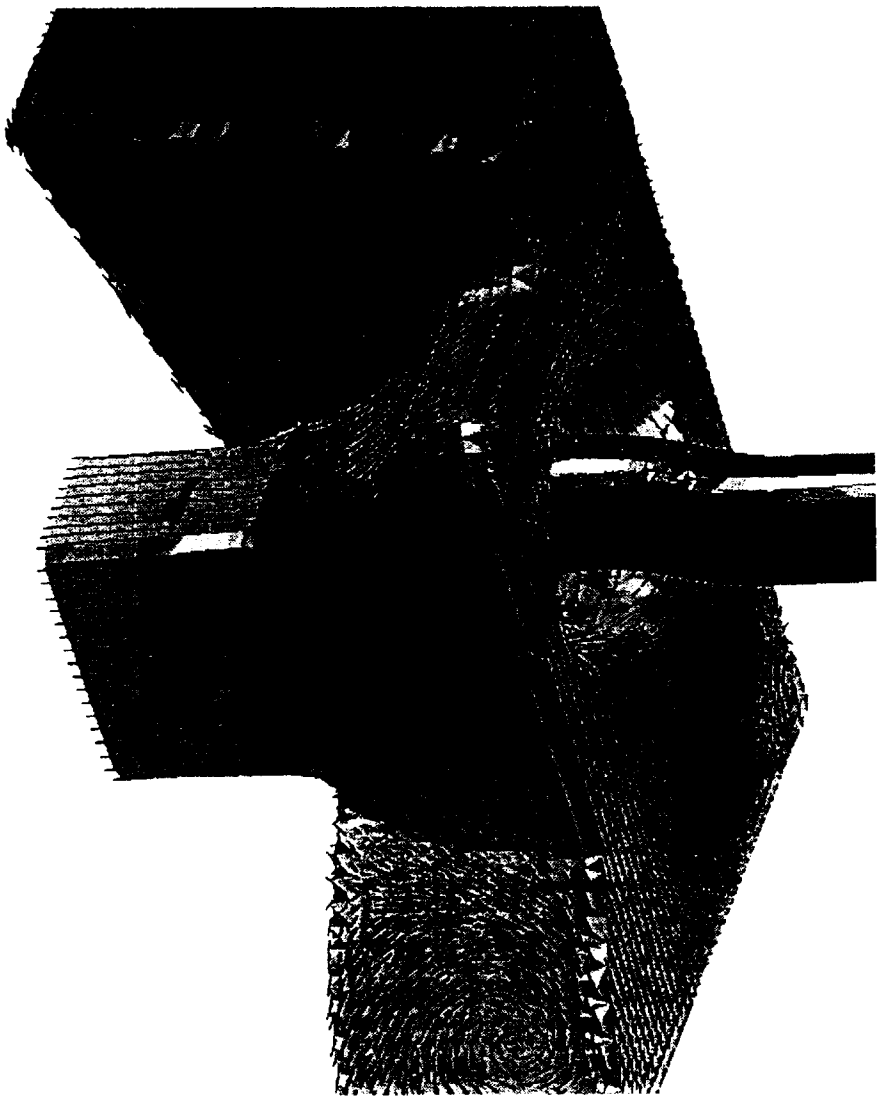


Figure 14. Helicopter Inlet Flow System Tufts

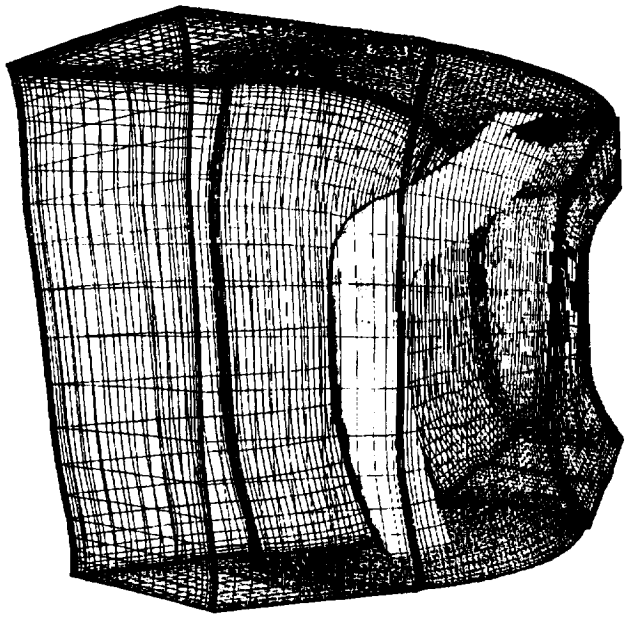


Figure 15. Probe Added to Compressor Grid

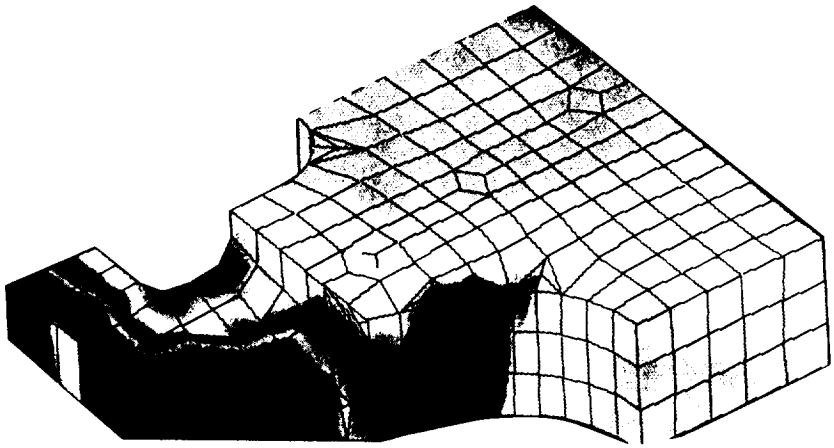


Figure 16. Structural Analysis of Bracket

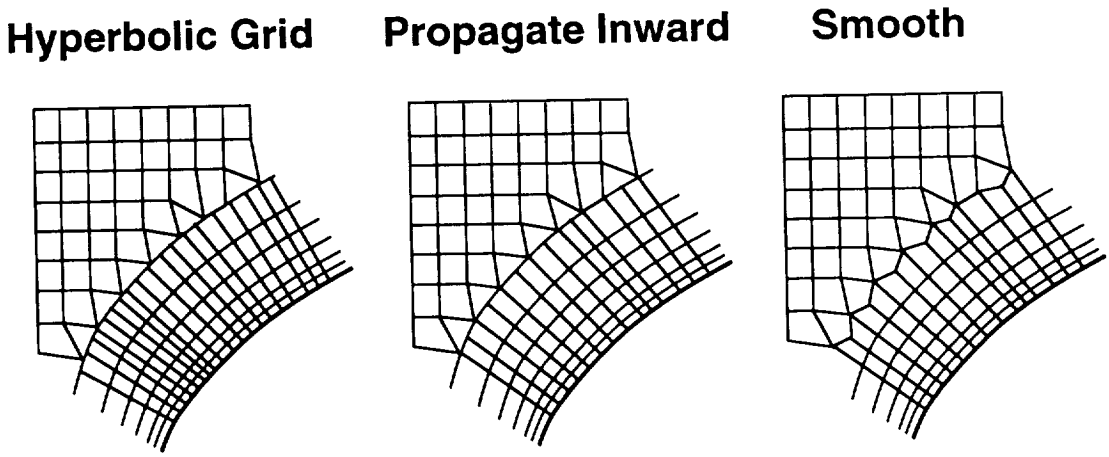



Figure 17. Viscous Grid Generation Process

# AUTOMATIC ZONING FOR STRUCTURED MULTI-BLOCK GRID

PRECEDING PAGE BLANK NOT FILMED

N95-28757

# ALGORITHMS FOR THE AUTOMATIC GENERATION OF 2-D STRUCTURED MULTI-BLOCK GRIDS

Thilo Schönfeld
CERFACS
42, avenue Gustave Coriolis
F-31057 Toulouse Cedex
France

Per Weinerfelt
Department of Mathematics
Linköping University
S-581 83 Linköping
Sweden

Carl B. Jenssen
Dep. of Industrial Mathematics
SINTEF – SIMa
N-7034 Trondheim
Norway

## SUMMARY

Two different approaches to the fully automatic generation of structured multi-block grids in two dimensions are presented. The work aims to simplify the user interactivity necessary for the definition of a multiple block grid topology. The first approach is based on an *advancing front method* commonly used for the generation of unstructured grids. The original algorithm has been modified toward the generation of large quadrilateral elements. The second method is based on the *divide-and-conquer* paradigm with the global domain recursively partitioned into sub-domains. For either method each of the resulting blocks is then meshed using transfinite interpolation and elliptic smoothing. The applicability of these methods to practical problems is demonstrated for typical geometries of fluid dynamics.

## INTRODUCTION

Structured multi-block grids are distinguished by regular quadrilateral cells (hexahedras in 3-D) which allow to obtain solutions of good accuracy. However, this high degree of regularity is too stiff when considering geometrical complex shapes. The direct mapping of the physical domain onto one *single* computational domain becomes increasingly complicated. The discretized domain must be partitioned into a set of sub-domains. For each of the resulting so-called blocks a grid is generated separately. The first step in the generation of such a multi-block grid is the definition of the block topology. An overall structure for the arrangement of the blocks, together with internal connectivity information and local coordinate orientations must be set up. Traditionally the block topology is set up manually, a pre-processing task that can quickly become difficult with rising complexity. This led us to the idea of replacing the user interactivity by an automatic generation of the blocks. Examples for references on techniques for the automatic meshing of quadrilateral grids are those of [2] or [8].

The aim of this work is to develop methods for generating structured multi-block meshes automatically in order to reduce the amount of pre-processing required for multi-block calculations. Multi-block solvers can then be applied to complex geometries with almost the same level of flexibility as solvers based on unstructured methods. Our original interest is to provide meshes for problems in fluid dynamics, but automatic mesh generators are highly demanded in many other fields.

The overall idea behind our first block-generation method is to combine the advantages of a structured multi-block grid with a technique commonly used to generate unstructured grids. In an unstructured environment the rigidity of the structured $(i, j)$-index system is broken and replaced by a system of pointers. Since the generation of an unstructured grid is done locally, and since only a 'single block' is required for a complex configuration, no special topology needs to be defined. We propose a block generation algorithm



PRECEDING PAGE BLANK NOT FILMED

that is based on the *advancing front technique* (AFT), which has been applied e.g. by Peraire et al. [6] or Löhner et al. [5] to generate small triangular elements (that directly form the final grid cells). Here, we use the advancing front technique to generate large elements that serve as blocks and thus have to be covered by mesh lines in order to obtain the final computational grid. The basic ideas behind the AFT algorithm are retained, with the exception of two principal modifications: Firstly, we create *quadrilaterals* instead of the originally generated triangles and, secondly, we generate *blocks* of maximal possible size rather than small *grid cells*. Compared to the basic AFT for triangles, the present algorithm for the generation of rectangular elements is more complicated, partly because of attempts made to ensure robustness and the prevention of distorted elements. The additional computational costs are easily justifiable, since much less elements are required compared to a complete unstructured mesh.

Earlier results with an application of AFT as a multi-block generator are presented in [1], where so-called 'micro-blocks' are created by forking triangles in an unstructured triangular mesh. However, this technique results in skewed blocks which degrades the accuracy of a flow calculation on the resulting meshes. It therefore seems advantageous to replace the triangular elements by the direct generation of quadrilateral blocks. This modified AFT-approach for the generation of quadrilaterals instead of triangles was first presented in [7] and is in the following referred to as 'method 1'. Although fully automatic, both of these AFT-approaches are based on underlying unstructured techniques. The drawback with these methods is that the topology is difficult to control which might lead to undesirable effects on the grid.

These shortcomings led us to the second approach ('method 2') [3], which, instead of building blocks as with AFT, is based on a successive partitioning of the domain of interest. This permits a better control of the topology. With this approach, the initial computational domain is successively sub-divided, until each sub-domain (block) is considered acceptable, according to certain criteria, for the generation of a structured mesh. The desired properties of such a block are for example that it should consist out of four corners where each corner should be as close as possible to a right angle. The partition procedure consists of first flagging boundary nodes as *cut required*, *cut permitted* or *cut not permitted* (corner node) and then chosing an optimal cut between two nodes. The success of this algorithm depends on the definition of the function measuring the quality of a cut. Though there are obviously several choices for this function, they all have to take into account various criteria such as the number of corners in the created sub-domains or the total number of boundary nodes still requiring a cut.

## METHOD 1: THE ADVANCING FRONT METHOD

The advancing front technique has obtained its name from a 'front' that 'advances' (travels) through the domain of discretization. This front, often referred to as the actual or current front, is defined as a closed curve of assembled line segments which changes continuously during the generation process. After each cycle of the algorithm one (or several) elements are generated, often by the simultaneous creation of a new grid point. A description of the underlying philosophy of our modified AFT-algorithm is given hereinafter.

### AFT for Rectangles

Background grid: The so-called background grid, used for interpolation purposes only, is created interactively and consists of large triangles which cover the complete domain of discretization (Fig. 1). At the vertices of each triangle, stretching parameters are stored. These values are user-defined before the generation process starts and enable the block size to vary throughout the domain. If desired, blocks of equal size can be obtained by using the same value at all the vertices. The information required at any point

in the domain is then obtained by linear interpolation of the stretching values at the vertices. In order to facilitate the creation of the background grid and to minimize the searching and interpolation procedures, the number of elements is kept as small as possible.

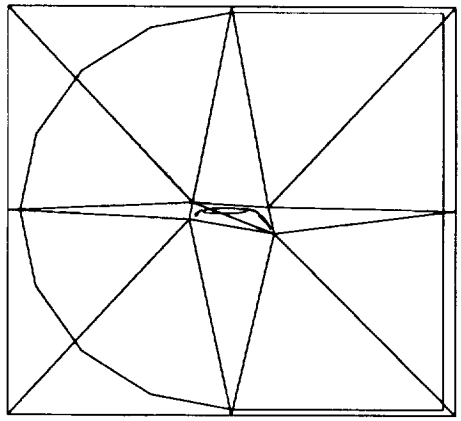


Figure 1: Background grid for multi-element airfoil.

Surface discretization: Unlike for the triangular AFT [6], in the present method the background grid is not employed for the discretization. Since large blocks are generated rather than the final mesh, a relative coarse approximation of the solid surfaces is sufficient.

In a standard application of the block generator, solid surfaces are discretized by equally-spaced points. These nodes belong to the final grid, and few of them are actually used in order to obtain a coarse discretization. In an alternative approach, all of these points are used to describe the geometry, thus leading to a much improved approximation of the solid surfaces. The supposed disadvantage of the latter technique for the purpose of the block generation is neutralized by the method of merging neighboring faces.

Initial front: By means of the background grid the starting front is set up. This consists of contiguous nodes on given discretized curves connected by straight lines. These line segments (in the following notation called 'faces') are referenced by integer arrays, with two indices describing the start and end-point of the face. In order to enable the use of one single closed curve (once the parts of the front that are initially separated are connected), the line segments on interior surfaces are stored in clockwise orientation (defined by the order in which the data is entered), while the faces on exterior boundaries are oriented in a counter-clockwise direction (or vice versa). The strict preservation of this rule throughout the entire generation process is an essential part of the algorithm.

## The Block Generation Algorithm

Compared to the generation algorithm for triangles, a large number of additional requirements are needed to control the automatic generation procedure. The robustness of the algorithm is one of the major considerations. The algorithm consists of the following principal steps, which are partially identical with those of the traditional triangle algorithm:

(i) As in the triangle algorithm, a choice is made for an initial base for the next element to be generated. The 'smallest length' criterion is used.

(ii) Next, the important means to merge faces comes into effect. The angle $\beta$ between the chosen initial basis and the faces next to it (on both ends) are checked. The initial base-line, together with the

faces that are to be merged, form the final basis with end-points $A$ and $B$ (Fig. 2). The pointers that indicate the positions of the adjacent nodes are then examined. In case $\overline{AB}$ belongs to a closed polygon with maximal six corners that cannot be sub-divided into two quadrilaterals, the items (iii) to (vi) listed below are skipped and the new element is set up directly.

(iii) The third major step is to find the position of the *two* ideal points $C_{0A}$ and $C_{0B}$. By 'ideal' we denote that point which is chosen if none of the nodes of the current front is selected. One ideal point for each the start and end-point of the basis has to be found. Since this task is more difficult compared to the standard AFT algorithm (with only one ideal node per element), below we give a more detailed description.

(iv) When the ideal points $C_{0A}$ and $C_{0B}$ have been selected, we determine all the potential nodes $C_i$ that belong to the actual front and which lie inside a circle of given radius $r$, with center $C_{0A}$ (reasonable value $r = 1.7 \times \delta_A$). For reasons of simplicity, we restrict our attention to the point $A$; the procedure for $B$ is analogous. The coordinates of the flagged points are ordered according to their distance from point $C_{0A}$; four 'reserve' nodes $C_R$ are added to the end of the list.

(v) In this step, the connecting point $C_A$ ($C_B$ for point $B$) is determined. This point must satisfy all of the requirements described below.

(vi) Once a valid point has been designated for both nodes $A$ and $B$, a new quadrilateral element is defined by the vertices $A$, $B$, $C_B$ and $C_A$. Four principal configurations are possible and the correct one has to be determined.

(vii) After the definition of a new element, the corresponding pointers are updated and the number of remaining faces is checked. If any faces remain, the whole procedure is repeated by starting at item (i); if none remain, the block generation is terminated.

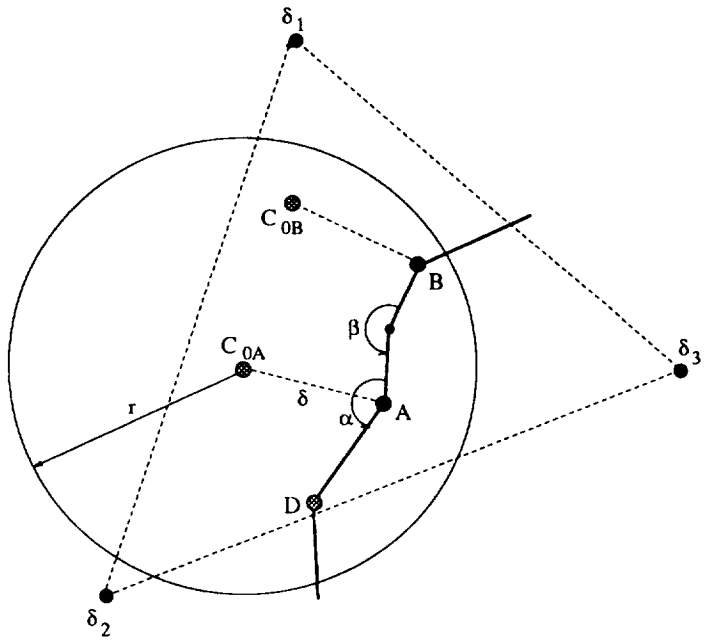



Figure 2: General notation for rectangle algorithm.

Face merging: This tool to reduce the number of blocks is one of the essential ingredients incorporated into our advancing front algorithm. Two neighboring faces may be merged to form one long face whenever the

angle $\beta$ between them is 'smooth', i.e. close to 180° (Fig. 3). The use of angles (which are non-dimensional) allows to circumvent the (heuristic) definition of the delicate threshold parameter that is necessary when working with the curvature (which depends on the geometric scaling of the configuration).

Typical values that define a smooth angle lie between 160° and 200°, although these threshold values may be chosen close to 180° at solid walls. Note that not only the base side may consist of several merged faces, but also the remaining three sides of each element. Indeed, the combination of faces opposite to the basis has been found to be efficient.

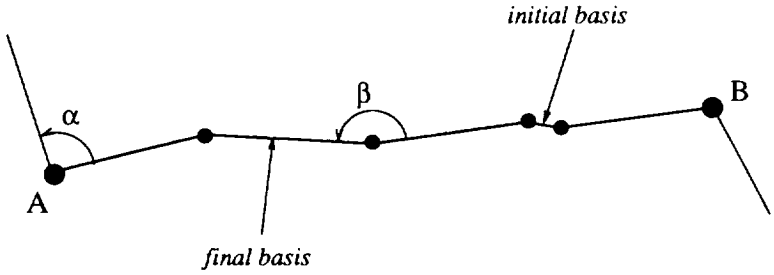



Figure 3: Face merging.

Finding the ideal points: For each end-point $A$ and $B$ of the final basis, two ideal points must be selected. Different principal strategies are used depending on whether the angle $\alpha$ between the base side $\overline{AB}$ and its neighbors is smaller or larger than 130°. This heuristic value, as well as all the other characteristic angles, has proved to be the most efficient when developing the program (efficient in terms of both algorithm robustness and general validity for any given configuration). We distinguish between the following cases (Fig. 4):

(i) $\alpha > 130°$ : The angle is large enough to be sub-divided without creating skewed elements. One new point is set up. The stretching length $\delta$ of the vector $\overrightarrow{AC_{0A}}$ is obtained from the background grid. The vector direction again depends on $\alpha$:

  (a) $130° < \alpha \leq 250°$: The angle is divided into two equal halves, i.e. the direction of $\overrightarrow{AC_{0A}}$ is the mean value of the normal directions to $\overline{AB}$ and $\overline{AD}$, where $D$ denotes the other node adjacent to $A$ on the present front.

  (b) $\alpha > 250°$: A sub-division into two halves would lead to skewed blocks. The adjacent face is then disregarded and $\overrightarrow{AC_{0A}}$ is chosen to be orthogonal to $\overline{AB}$.

(ii) $\alpha \leq 130°$ : Again, the subdivision into two halves would lead to an element with a distorted shape. The nearest neighbor of $A$ is chosen as an ideal point; the length of $\overline{AD}$ is a given length $\delta$.

Since we generate quadrilateral elements, no limitation on the stretching length $\delta$ is necessary (contrary to the classical AFT). Long stretched blocks are allowed.

Criteria for point validation: In this section we give a survey of the criteria to be satisfied by a point $C_i$ in order to be a valid node $C_A$.

(i) The point under consideration $C_A$ does not belong to any of the merged faces that form the base edge.

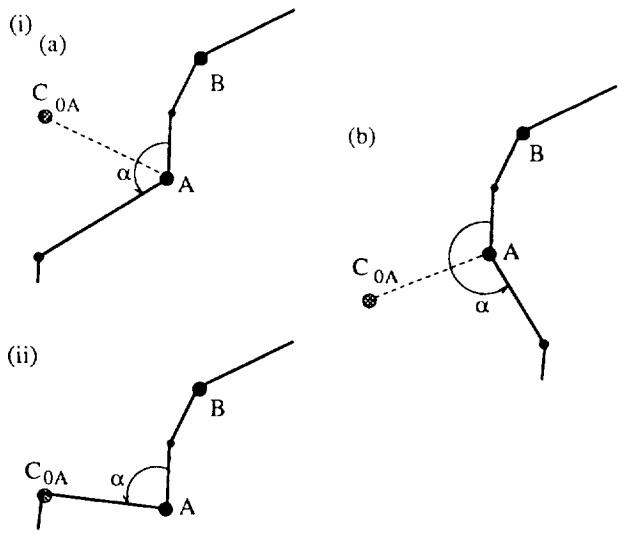



Figure 4: Checking of angles.

(ii) In the case when the point $C'_A$ is one of the neighboring nodes, the angle $\alpha$ must not be too large. This check is performed in a successive way for all the contiguous points of $A$.

(iii) The angle between $\overline{AB}$ and $\overline{AC'_A}$ obeys $50° \leq \angle BAC'_A \leq 130°$.

(iv) The triangle $\triangle BAC'_A$ has a positive area, i.e point $C'_A$ lies in the interior of the domain. Here the strict maintenance of the orientation plays an important role. This condition corresponds to the first requirement of the standard AFT algorithm.

(v) The second well-known criterion of the classical AFT approach checks for any intersections of the line $\overline{AC'_A}$ with existing faces. Note that crossings with *all* existing faces must be examined, not just those with edges on the *current* front.

These tests are carried out for the flagged points of both end-nodes $A$ and $B$ of the basis. The list of reserve points guarantees that at least one node fulfills all of these five conditions. Since the check for node $B$ follows that for $A$, the designated node $C'_A$ is already known. Additional validity requirements must be imposed on point $C'_B$:

(vi) Trivial: Node $C'_B$ must not be identical with $C'_A$.

(vii) There must be no intersections between the potential line $\overline{C'_A C'_B}$ and any of the existing edges.

Creation of a new element: During one 'quadrangulation' step exactly *one* new element is generated. This characteristic feature of the algorithm maintains a clear structure for the generation process. The type of this element is determined by the kind of the new nodes $C'_A$ and $C'_B$. In the following, we describe the principal configurations that can appear (see Fig. 5):

(i) In the ideal case, one new node is placed in the domain for either node $A$ and $B$. Three new faces are formed. Points $C'_A$ and $C'_B$ are identical with $C'_{0A}$ and $C'_{0B}$, respectively (or one of the reserve nodes).



ORIGINAL PAGE IS
OF POOR QUALITY

(ii) If a neighbor of $A$ (or $B$) is found, a new point is chosen for $B$ (or $A$). One new node and two faces are created. The faces that form the side $\overline{AC}_A$ are deleted from the actual front. In the special case when an adjacent node has been chosen for both $A$ and $B$, no new point is set up and the connection line $\overline{C_A C_B}$ forms one new side.

(iii) If (as in the previous case) one neighbor of $A$ has been chosen, but the point $C_B$ is located on another part of the front, no new node is set up, and one new side is created. All the faces that form $\overline{AC}_A$ are removed from the current front, as well as the faces between $C_A$ and $C_B$. Similarly for point $B$. In the special case when both $C_A$ and $C_B$ are located on another part of the front, but neither is a neighbor of either $A$ or $B$, no new node but two new sides are defined.

(iv) In the final case, an element with either quadrilateral, pentagonal, or hexagonal shape is chosen. The reason for the implementation of this special case was to reduce the computational costs by avoiding the node validation verifications. Starting at the base side, all the faces that form the polygon are discarded.

For all of the cases mentioned above, the faces which form the base side are removed from the list of actual front faces. The new faces are added to the current front, hereby conserving the strict orientation rule.

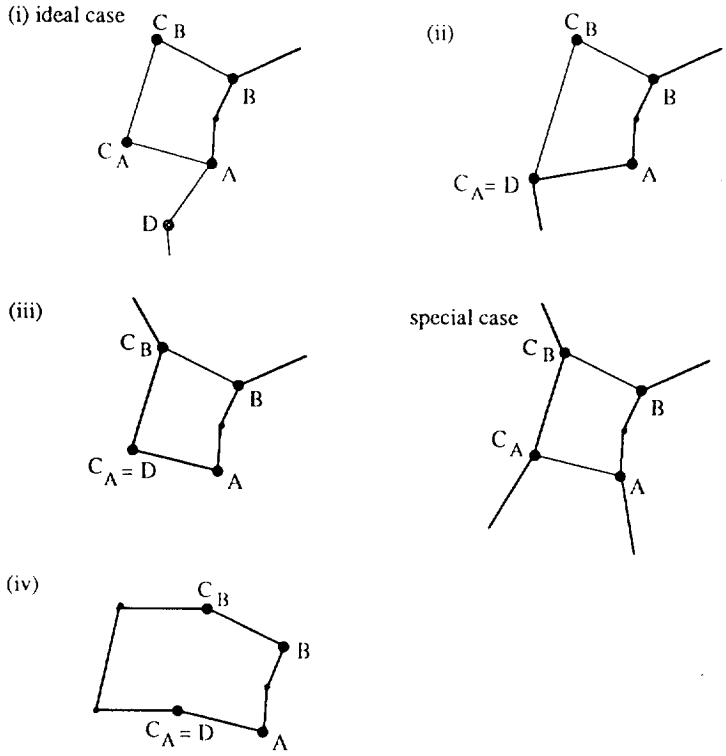



Figure 5: Potential new elements.

## The Pointer System

The data structure of the present block generator employs a pointer system. Two pointers are necessary to keep track of the block generation process. The first relates the nodes to the element faces. For each face, this array contains: in its first position, the index of the start point; in the second position,



ORIGINAL PAGE IS
OF POOR QUALITY

the end-point; in a third position, the face boundary condition type is stored. A second pointer contains the inverse information and links the faces to the nodes (though this is no longer of importance when the corresponding face is deleted from the actual front).

Additional pointers are required to provide the necessary information for both the mesh generator and the flow solver (note that these pointers are independent of the block generation procedure and thus are not essential for the algorithm). An element-to-face pointer relates the faces to the four block sides. In a strict rule faces 1,2 and 3,4 are pairs of opposite faces of each element. The inverse pointer links the elements to the faces. The third output pointer is the face-to-node pointer. Finally, the fourth pointer stores the number of merged points on solid wall faces (this information is necessary for the mesh generator).

### Block Generator Post-Processing

The automatic generation procedure usually gives a larger number of blocks than desired, since the robustness of the algorithm is deemed to be more important than creating as few elements as possible. For this reason, a post-processing routine has been developed. After viewing the data given by the block generator, neighboring blocks may be merged manually. The second principal purpose of post-processing is to initialize the number of grid points per block-side.

## METHOD 2: DIVIDE AND CONQUER

The second of our methods has been developed with the experience gained by the previous approach and is based on a *divide-and-conquer* paradigm.

### Block Generation by Successive Domain Partitioning

The basic idea behind this technique is to successively partition a given domain into sub-domains until all sub-domains are acceptable for mesh generation, according to certain criteria. This principle is illustrated by the recursive divide-and conquer block creation procedure, **createblock**, written in a pseudo language:

```
procedure createblock(domain);
    if (domain not a block)
% partition the domain into subdomain 1 and subdomain 2
        partition(domain, subdomain 1, subdomain 2);
        createblock(subdomain 1);
        createblock(subdomain 2);
    end
end procedure
```

As can be seen from the code, we need a definition of a *block* and a *partition* procedure. The criteria defining a block should ideally be based on mathematical principles related to eg. the shape of the domain. Since it might be difficult to find a proper formulation, we have instead chosen heuristic criteria mimicking those used in an interactive block generation session. The desired properties of a block are hence:



ORIGINAL PAGE IS
OF POOR QUALITY

(i) A block should have four corners.

(ii) Each corner should be as close to a right angle as possible.

(iii) Each side should have a limited curvature.

(iv) The area of the block should be as large as possible.

The main reason for considering these is that they support the generation of high quality meshes by simple mesh generation algorithms like transfinite interpolation and elliptic smoothing. We need, however, to modify items (i)-(iv) slightly in order to obtain robust criteria for the block partitioning. Here, robust means that the algorithm should terminate after a finite number of steps, i.e. any given domain should be split into a finite number of blocks. Criteria (i)-(iv) are also involved in the procedure of deciding if and how to subdivide a given domain, i.e. the main part of the partition procedure. Before formulating the final block criteria and the partition procedure, we start by introducing some notations and definitions.

The input data to the createblock routine is a 2–D domain with a boundary which is a closed polygonal curve with positive orientation as shown in Fig. 6.

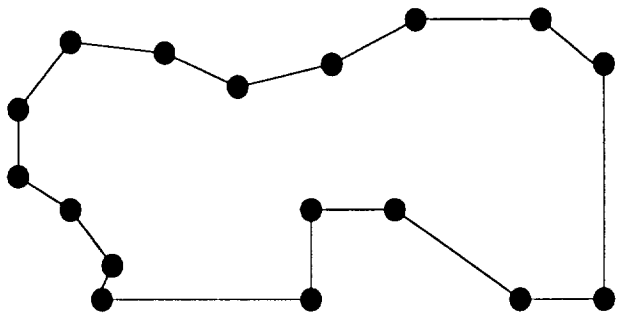


Figure 6: Sample domain.

First all vertices of the polygon, i.e. boundary nodes, are flagged. The flagging options are *cut required* (*cr*), *cut permitted* (*cp*) and *cut not permitted* (*cnp*). The first and foremost criteria when flagging the nodes is the size of the angle $\alpha$ between two neighboring line segments as shown in Fig. 7.

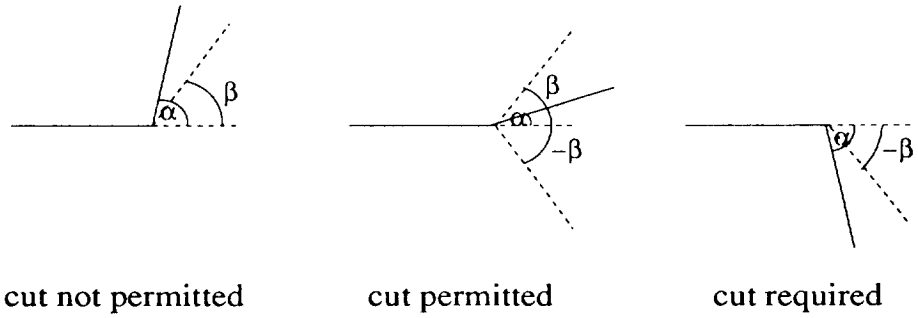



Figure 7: Flagging of nodes based on the angle.

Let $\beta$ be a predefined fixed angle, $0 \leq \beta \leq \pi/2$, then a node is marked as *cr* if $\alpha$ fulfills $\alpha \leq -(\frac{\pi}{2} - \beta)$. If $-(\frac{\pi}{2} - \beta) < \alpha < \frac{\pi}{2} - \beta$ the node is marked as *cp*. Finally, when $\alpha \geq \frac{\pi}{2} - \beta$ the node is marked as *cnp*. A *cnp* node is also called a corner. Furthermore we need to introduce the notation *accumulated curvature*. The accumulated curvature between two corners is defined as the sum of the absolute value of the angles $\alpha$. The reason for using this notation is that property (iii) above suggests a limitation of the curvature of a block side. Hence the accumulated curvature is computed for all nodes along the boundary and each time it exceeds a multiple of a predefined angle $\gamma$ the actual node is marked as *cr* (see Fig. 8).

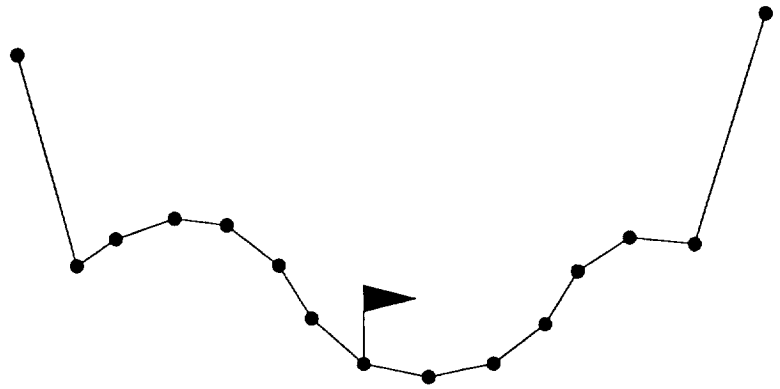


Figure 8: Flagging of nodes based on the accumulated curvature.

We can now proceed defining the modified criteria of a block. As mentioned above the final blocks should ideally have the desired properties (i)-(iv). Some of these are included in the partitioning algorithm, as will be seen later, and hence not be applied explicitly. The following criteria for a block are finally chosen:

(i) A block has 3, 4, or 5 corners.

(ii) A block does not contain any node flagged as *cr*.

The first requirement ensures that the blocking procedure terminates after a finite number of steps. This might not be the general case if we only allow blocks with four corners. As illustrated in Fig. 10 triangles and pentagons cannot be partitioned into quadrilaterals with a single cut, without splitting the corners, since splitting such blocks result in at least one new block with same number of corners. Thus obtaining a domain with three or five corners has to be accepted as a block in order to terminate the splitting of the domain. If a domain is not accepted as a block, it is passed to the partition procedure for further subdivision. We will now continue the description of the partition procedure. First, a set of points, between which a cut can be performed, has to be prescribed. The cut will here be restricted to the set of boundary nodes. Among all possible cuts only *allowed cuts* are considered. By an 'allowed' cut we mean a cut between a node marked as *cr* and a *cr* or *cp* node. The *best cut* is finally selected from the set of allowed cuts. Hence a domain, which is not a block, need to have at least one *cr* node in order to be split. If no such node exists the node corresponding to the smallest angle $\alpha$ is re-flagged to a *cr* node. Finally, additional boundary nodes, flagged as *cp*, are inserted to ensure an ample amount of possible cuts to chose from. These nodes are chosen such that no line segment is longer than a predefined characteristic length, $l_{scale}$, and so that there is at least one *cp* node between two *cnp* nodes.

We now proceed, by describing how to select the best cut among all legitimate cuts. The success of the algorithm depends crucially on the definition of best cut. Following the approach of Talbert and

Parkinsson [9], we define 'best cut' as that cut which minimizes a function measuring the quality of the two new domains created by the cut. This function is chosen as a linear combination of terms measuring the angles between the cut and the boundary, the distance from the cut to the boundary, the number of corners in the resulting subdomains, and whether or not both nodes defining the cut are flagged as *cr*. Thus our function $f$ is defined as:

$$f = w_1\theta + w_2 s + w_3 n_c + w_4 r \tag{1}$$

where $w_1$, $w_2$, $w_3$, and $w_4$ are non-negative weights and

$$\theta = \min_{n=-1,0,1} |\alpha/\pi - n/2| \tag{2}$$

$$s = \max(0, (l_{scale} - l)/l_{scale}) \tag{3}$$

$$n_c = \begin{cases} 1 & \text{if number of corners} = 3 \text{ or } 5 \text{ in either of the two blocks} \\ 0 & \text{otherwise} \end{cases} \tag{4}$$

$$r = \begin{cases} 1 & \text{if only one node is flagged } cr \\ 0 & \text{if both nodes are flagged } cr \end{cases} \tag{5}$$

The angle measure (2) tends to encourage cuts that intersects the boundary either at an approximately right angle, or that makes a smooth transition from boundary to cut. For clearity, this is illustrated in Fig. 9.

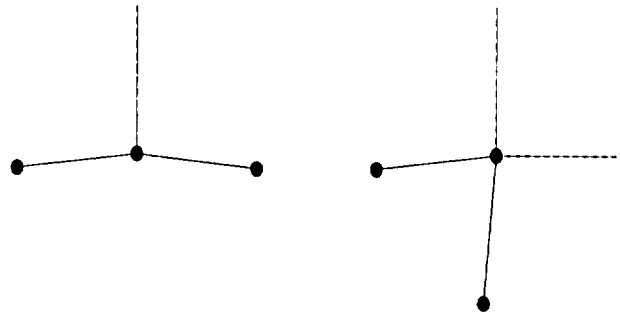


Figure 9: Preferred angles between boundary and cut.

The distance measure (3) is designed to avoid excessively thin blocks. Here the parameter $l_{scale}$ is a typical length scale. The corner count (4) has the purpose of avoiding domains with three or five corners. The final measure (5) encourages cuts between two nodes that both are flagged as *cr*. This is intended to reduce the total number of cuts that has to be performed.

Finally, before the generated block topology can be passed to the mesh generator, it must be post-processed so that all blocks have four sides. This is achieved by a re-flagging of nodes wherever necessary. This re-flagging is again based on the angle between neighboring line segments as shown in Fig. 7. For

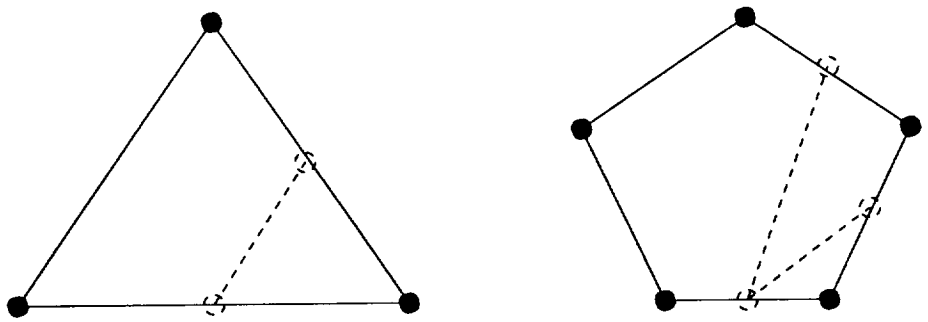


Figure 10: Subdivision of triangles and pentagons.

pentagons the corner node with the smallest angle is ignored, while for triangles, the node with the largest angle not already defined as a corner, is flagged as a corner. Note, that only the definition of corners or sides changes, while the actual shape of the block remains unchanged.

The output from the block generator then consists of blocks as shown in Fig. 11. Each block has four sides, where each can be sub-divided into a number of edges. The different edges correspond to different boundary conditions. All boundary conditions, except internal boundaries (i.e. boundaries which are shared by two blocks), are supplied by the input geometry. For internal boundaries the condition is defined simultaneously with the creation of new internal edges.

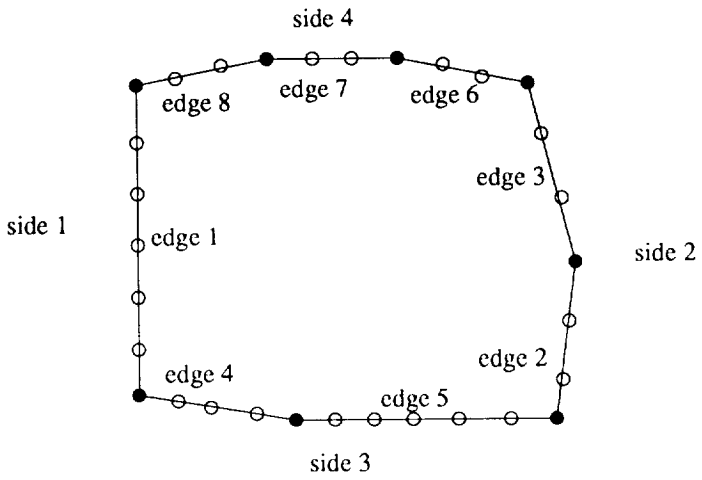



Figure 11: Information created by the block generator.

## MULTI-BLOCK MESH GENERATION

The second part in the multi-block generation method consists of generating a structured mesh in each block. This part is common to both of the methods described above and the same mesh generation tool is used. Before a mesh can be generated, additional information has to be created by a block-grid generator interface.

## The Block–Grid Generator Interface

The block generator provides topology information in the form of node-coordinate, edge-node, and edge-side-block registers together with the type of boundary condition on each edge. From these pointers all necessary information for the grid generator can be obtained. Part of the data can be used directly while others are retrieved from an interface program.

Before generating the mesh the number and the type of boundary conditions, the connectivity to other blocks and the number of grid points on each of the four block sides must be determined. The crucial task is to compute the number of grid points on the edges from which we get the number on the block sides. If the blocks are meshed completely independent of its neighbors this is trivial. Then for each block an arbitrary number of points can be specified in the two local coordinate directions. This, however, results in blocks with discontinuous grid lines at the boundaries. We prefer instead to work with patched blocks with coincident grid lines. This means that the number of points on each edge, under the constrains described below, have to be computed. This is, for an arbitrary block topology, a non-trivial task. After some attempts to do this interactively, we decided instead to formulate the problem in a mathematical framework, as a optimization problem, and solve it by a well known technique.

To start with, let $n_i$ denote the number of points on edge number $i$. For each block the following equations hold:

$$\begin{aligned} \textstyle\sum_{i\in E_{1,b}}(n_i-1) &= \textstyle\sum_{i\in E_{2,b}}(n_i-1) \\ \textstyle\sum_{i\in E_{3,b}}(n_i-1) &= \textstyle\sum_{i\in E_{4,b}}(n_i-1) \end{aligned} \tag{6}$$

where $E_{k,b}$ is the set of edges belonging to side $k$ in block $b$. These are the compatibility equations which guarantee the same number of grid points on the corresponding block sides $1,2$ and $3,4$. For a solid wall edge $n_i$ is governed by the number of points, $N_i$, in the geometry description. To ensure that $n_i$ is not less than $N_i$ we impose the constraint $N_i \le n_i$. For the remaining edges we need lower and upper bounds, $L_i$ and $U_i$ on $n_i$ i.e. $L_i \le n_i \le U_i$. The numbers $L_i$ and $U_i$ can be specified directly by the user or generated from a given grid point density function. The problem is now to find a solution to the underdetermined system (6) with the constraints above. Among all feasible solutions we want a solution which does not lead to unnecessary many grid points. Hence, it is reasonable to look for the solution which minimizes the total sum (or a weighted sum) of grid points. The final mathematical formulation then reads:

$$\begin{aligned} &\min_{n_i} \textstyle\sum_{i\in E} n_i \\ &\textstyle\sum_{i\in E_{1,b}}(n_i-1) = \textstyle\sum_{i\in E_{2,b}}(n_i-1) \\ &\textstyle\sum_{i\in E_{3,b}}(n_i-1) = \textstyle\sum_{i\in E_{4,b}}(n_i-1) \quad \text{for all blocks} \\ &n_i \ge N_i \quad i \quad \text{solid wall face,} \\ &L_i \le n_i \le U_i \quad i \quad \text{not solid wall face,} \quad n_i \quad \textit{integer} \end{aligned}$$

This is a linear integer programming problem which is solved by a standard technique based on branch and bounds.

If a global smoothing of the mesh is desired a search for singular points is also necessary. All the singular nodes are stored in an array together with their neighbor points and treated in a special way by the smoother as described in the next section. The block data structure used by the elliptic smoother is the same as for a multi-block flow solver which means that flow calculations can be done immediately when the mesh and the block data are available.

## Grid Generation

The output from the mesh generator is a structured grid in each block, with coincident grid lines on the interfaces between two blocks. As mentioned in the previous section, an alternative approach, not considered in this work, is to allow the number and position of these grid points to vary independently in each block. This technique may prevent grid spreading, but needs a more sophisticated flow solver which includes conservative interpolation routines.

The initial mesh is generated by linear or hermitian transfinite interpolation. This gives often a good start grid due to the feature of the topology generator which provides blocks with rectangular shape and without strong curvature. The mesh quality is then further improved. The mesh is first stretched in the direction normal to solid wall boundaries. The resulting mesh will then better resolve the flow in these areas. The mesh is then smoothed by solving the well known elliptic equations (cf. Thompson et al. [11]):

$$\begin{cases} \alpha x_{\xi\xi} - 2\beta x_{\xi\eta} + \gamma x_{\eta\eta} &= -J^2 (x_\xi P + x_\eta Q) \\ \alpha y_{\xi\xi} - 2\beta y_{\xi\eta} + \gamma y_{\eta\eta} &= -J^2 (y_\xi P + y_\eta Q) \end{cases} \tag{7}$$

where $(x, y)$ and $(\xi, \eta)$ are coordinates in the physical and computational plane respectively, $\alpha = x_\eta^2 + y_\eta^2$, $\beta = x_\xi x_\eta + y_\xi y_\eta$, $\gamma = x_\xi^2 + y_\xi^2$ and $J = x_\xi y_\eta - x_\eta y_\xi$.

Following the technique suggested by Thomas and Middlecoff [10], the source terms $P$ and $Q$ are computed on the block boundaries and extended to the interior by linear interpolation.

$$\begin{cases} P = -(x_\xi x_{\xi\xi} + y_\xi y_{\xi\xi}) / \left(x_\xi^2 + y_\xi^2\right) & \text{along the boundaries} \quad \eta = \text{constant} \\ Q = -(x_\eta x_{\eta\eta} + y_\eta y_{\eta\eta}) / \left(x_\eta^2 + y_\eta^2\right) & \text{along the boundaries} \quad \xi = \text{constant} \end{cases} \tag{8}$$

Hence, the distribution of the boundary points spreads into the interior of the block. Finally, system (7) is discretized and solved iteratively by means of the Jacobi method. We are mainly interested in applying this technique in order to smooth the grid. If, however, there is a need for solving the equations completely, a multi-grid technique can be applied as an option, in order to speed up the convergence.

A special smoothing procedure is used for the singular points. Here, the mesh is smoothed iteratively by a discrete version of the undivided Laplacian operator:

$$\vec{r}_k^{n+1} = \vec{r}_k^{n} - \omega \sum_{i=1}^{N_p} (\vec{r}_k - \vec{r}_i)$$

where superscript $n$ denotes the iteration counter, $\vec{r}_k$ is the singular point, $\vec{r}_i$, $i = 1, ..., N_p$ its neighbors and $\omega$ a relaxation parameter ($0 \leq \omega \leq 1$).

## RESULTS - COMPARISON OF METHODS

The two algorithms for the automatic generation of multi-block topologys are presented with the help of four different configurations. The generic configuration of a collection of four circles of different radius provides a first challenging test for the second of our methods. In Fig. 12 the global topology with 16 blocks and the final mesh are shown. The overall grid is characterized by a few non-smooth cells at the intersection of the blocks. Here singular points are created, which require special attention by the flow solver algorithm.

The double profile represents the first test case for method 1. Fig. 13 shows the global mesh topology consisting of 36 blocks. The overall topology type for this airfoil-flap configuration is a C-mesh, which is fairly well followed by the block structure. A close-up of this block topology together with the final mesh is given in Fig. 14. We note the smooth distribution of grid points (though the overall mesh is very coarse).

Two-dimensional mesh generation can also be used for axi-symmetric problems. As an example, we have meshed a valve-cylinder assembly which represents the first case to which both of our methods are applied. The blocks generated for this simplified internal flow configuration are depicted in Fig. 15. Method 1 generates nine original blocks (where some of them could be merged afterwards), while method 2 creates six blocks. While the shapes of the blocks in the valve part are similar, their topology in the cylinder part differs substantially. This difference, although not crucial, is transmitted to the meshes (Fig. 16).

A high-lift multi-element airfoil configuration represents the final test case. This geometry is the most difficult among our configurations due to the presence of several different length scales as well as strong curvature on the boundary. The present configuration is characterized by a main profile, a slat and two flaps that render a structured single-block generation quite impossible. The global C-type block topology for method 1 (75 blocks) and method 2 (21 blocks) is displayed in Fig. 17. The two approaches give substantially different arrangements of the blocks. With the divide–and–conquer method large blocks are generated that can extend from the surface of the airfoils to the far-field boundary of the domain. These large blocks are not obtained with the AFT-algorithm, which creates by far more blocks (where, again, some of them can be merged afterwards). The global meshes are displayed in Fig. 18. The underlying block topology becomes evident. Again, as with the generic circle configuration, with method 2 singular points can occur at block intersections. For a better look at the geometry a close-up of these block distributions is given in Fig. 19. We note the good clustering of a small number of blocks for method 2 in the region close to the airfoils. A similar view in Fig. 20 shows the computational grids. Finally, in Fig. 21 the solution of the Euler equations using the cell-centered multi-block solver of [4] on the mesh generated by method 2 is given. The figure shows the computed Mach number field for a free-stream Mach number of 0.15 and 10° angle of attack.

## CONCLUDING REMARKS

Two different approaches for the automatic generation of the block topology of structured multi-block

grids are presented. Both methods have been successfully applied to different geometries.

The first approach is based on an advancing front method. Compared to the original algorithm for triangles, the present method for the generation of rectangular elements is more complicated, partly because of attempts made to ensure robustness and the prevention of distorted elements. The additional computational costs are easily justifiable, since much less elements are created than for the generation of a completely unstructured mesh.

One of the major criticism leveled at the AFT is the background grid, which prevents the advancing front technique from being entirely automatic. A certain amount of user experience (or trial and error labour) is required to first place the vertices of the background grid triangles at the right locations and next to allocate suitable stretching parameters to these nodes. In our opinion the background grid needs to be replaced by a more automatic method, which provides the information about the element size. This information could for example be obtained from the curvature of the surface description. Further, robustness of the algorithm is not always guaranteed, due to the heuristics involved in keeping the algorithm stable and to enable its general application for a wide a range of configurations. Blocks need to be merged afterwards, both in order to reduce their total number and to avoid extrem differences in the block size. It is desirable to have a better control over the topology type of the global grid.

The second of our procedures is based on a divide-and-conquer principal. The algorithm basically consists of first flagging boundary nodes as *cut not permitted* (i.e. a corner), *cut required* or *cut permitted*, and then chosing an optimal cut between two nodes. The success of the algorithm depends on the definition of the function measuring the quality a cut. Obviously, a number of adequate functions can be constructed, and we do not claim having found the best one. It is however our experience that this function has to take into account the angles of the intersection of the cut with the boundary, the distance from the cut to the boundaries, the number of corners in the created subdomains, as well as the total number of boundary points still requiring a cut. The non-trivial problem of specifying the number of grid points on each block boundary can be formulated as a linear integer programming problem. With this approach, the only information that has to be supplied by the user is a lower bound for the number of points on a boundary edge.

A comparison of these two methods reveals better performances achieved with the divide-and-conquer approach. A smaller number of blocks is generated than with the AFT-method. The better control over the overall topology allows to cluster blocks in regions of complicated shapes, while at the same time only a few blocks are created away from the body surfaces. However, the problem of singular nodes still needs to be resolved. A future continuation of this work would therefore most likely be based on method 2. The basic divide-and-conquer principal applies directly to three-dimensional problems for realistic configurations. Modifications of the 2-D algorithms would comprise aspects such as the current way of flagging nodes and performing the cuts.

## REFERENCES


1. Bergman, C.M.: *Developpement de Methodes Numeriques pour des Ecoulements Hypersoniques Non-visquex Autour d'Engins Spatiaux*, PhD thesis, Institut National Polytechnique de Toulouse, 1990.

2. Blacker, T.D. and Stephenson, M.B.: Paving: A New Approach to Automated Quadrilateral Mesh Generation, Intern. Journal for Num. Meth. in Engineering, 32, pp. 811-847, 1991.

3. Jenssen, C.B. and P. Weinerfelt: Automatic Multi-Block Mesh Generation in Two Dimensions, SINTEF

Industrial Mathematics, Report STF10 A93007, Trondheim, 1993.

4. Jenssen, C.B.: *Implicit Multi Block Euler and Navier-Stokes Calculations*, AIAA Journal, 32, No 9, pp. 1808–1814, 1994.

5. Löhner, R. and Parikh, P.: Generation of Three-Dimensional Unstructured Grids by the Advancing Front Method, AIAA-Paper 88-0515, Reno, January 1988.

6. Peraire, J. et al.: Adaptive Remeshing for Compressible Flow Computations, Journal of Comp. Phys. 72, pp. 449-466, 1987.

7. Schönfeld, T. and P. Weinerfelt, P.: The Automatic Generation of Quadrilateral Multi-Block Grids by the Advancing Front Technique, in Proceedings of the *Third International Conference on Numerical Grid Generation* in Barcelona, pp. 743-754, A. S.-Arcilla, J. Häuser, P.R. Eiseman, and J.F. Thompson (Eds.), Elsevier Science Publishers, North Holland, June 1991.

8. Steinthorsson, E. and Ameri, A.A.: Computations of Viscous Flows in Complex Geometries using Multiblock Grid Systems, AIAA-Paper 95-0177, Reno, January 1995.

9. Talbert, J.A. and Parkinsson, A.R.: Development of an Automatic Two-Dimensional Finite Element Mesh Generator using Quadrilateral Elements and Bezier Curve Boundary Definition, in *International Journal for Numerical Methods in Engineering*, 29, 1990.

10. Thomas,P.D. and Middlecoff, J.F.: Direct Control of the Grid Point Distribution in Meshes Generated by Elliptic Equations, AIAA Journal, 18, 1980.

11. Thompson, J.F., Warsi, Z.U.A. and Mastin, C.W.: *Numerical Grid Generation: Foundations and Applications*, North-Holland, 1985.

## FIGURES

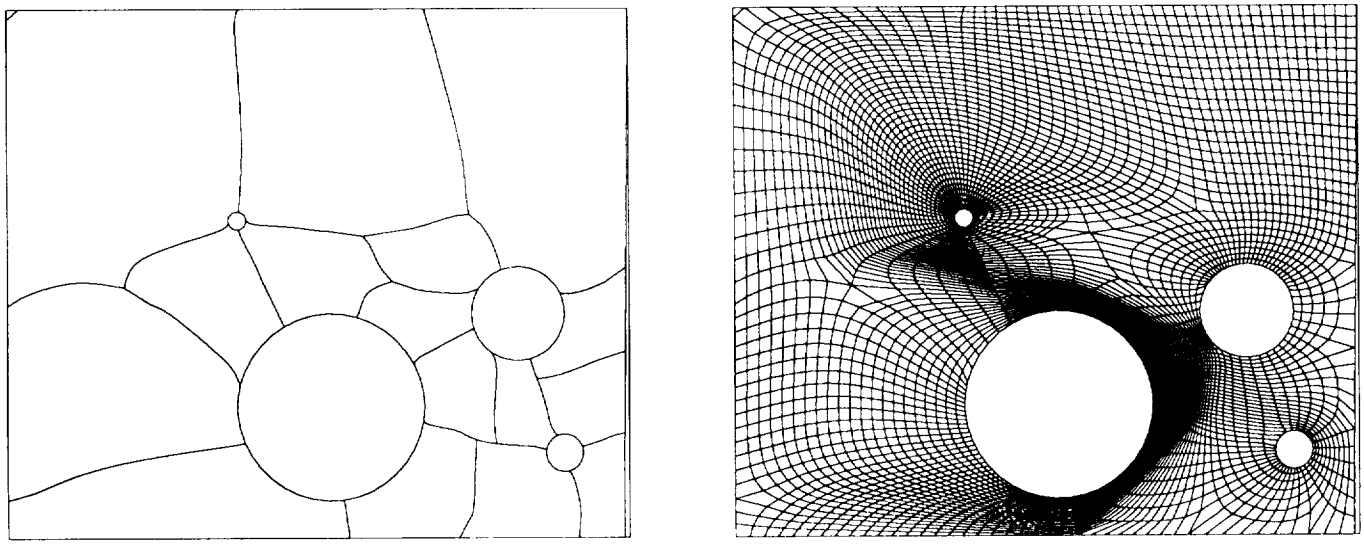


Figure 12: Global block topology (left) and final mesh for 4-circle configuration (method 2).

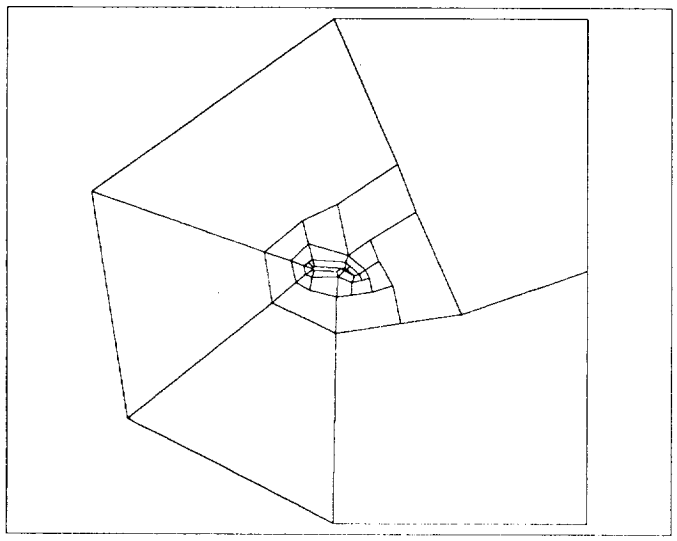


Figure 13: Global topology with 36 blocks for double profile (method 1).

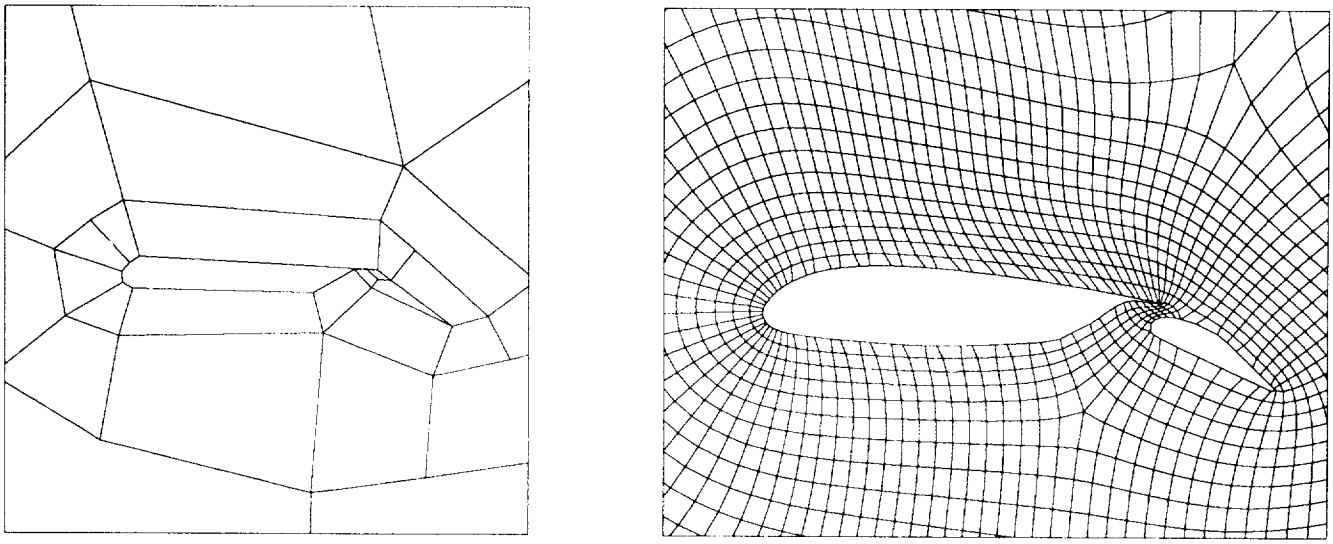


Figure 14: Close-up of block topology (left) and final mesh for double profile (method 1).

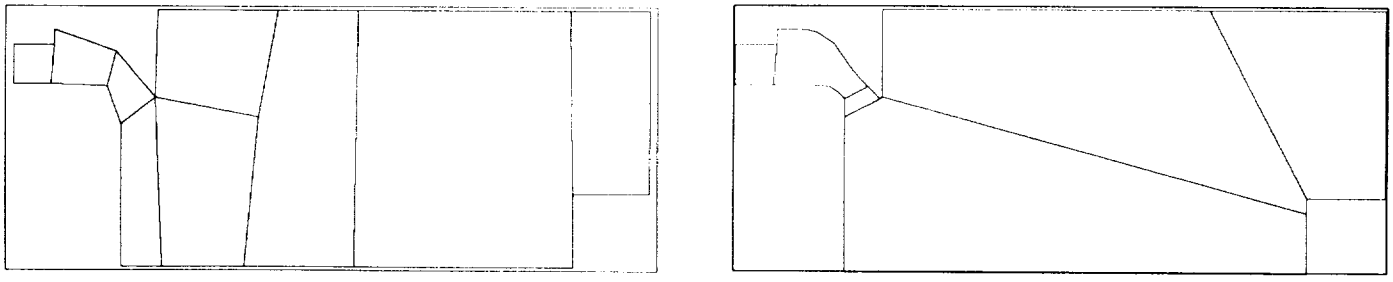


Figure 15: Valve-cylinder: global block topology for method 1 (left) and method 2.

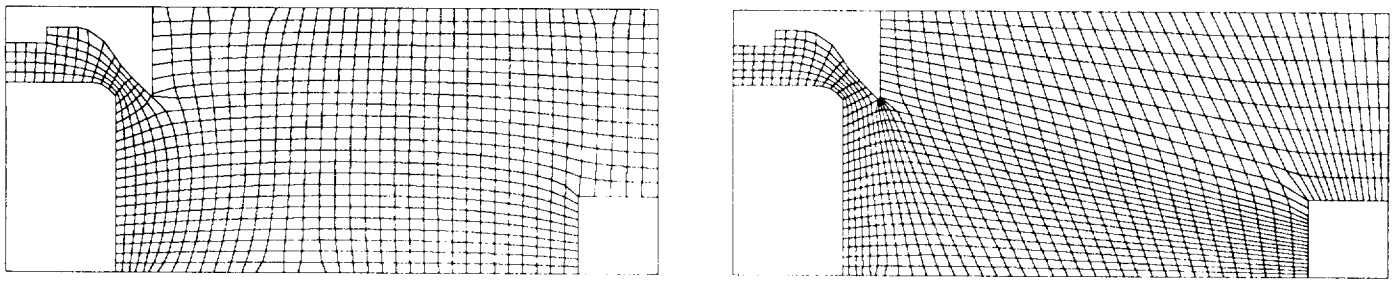


Figure 16: Valve-cylinder: final mesh for method 1 (left) and method 2.

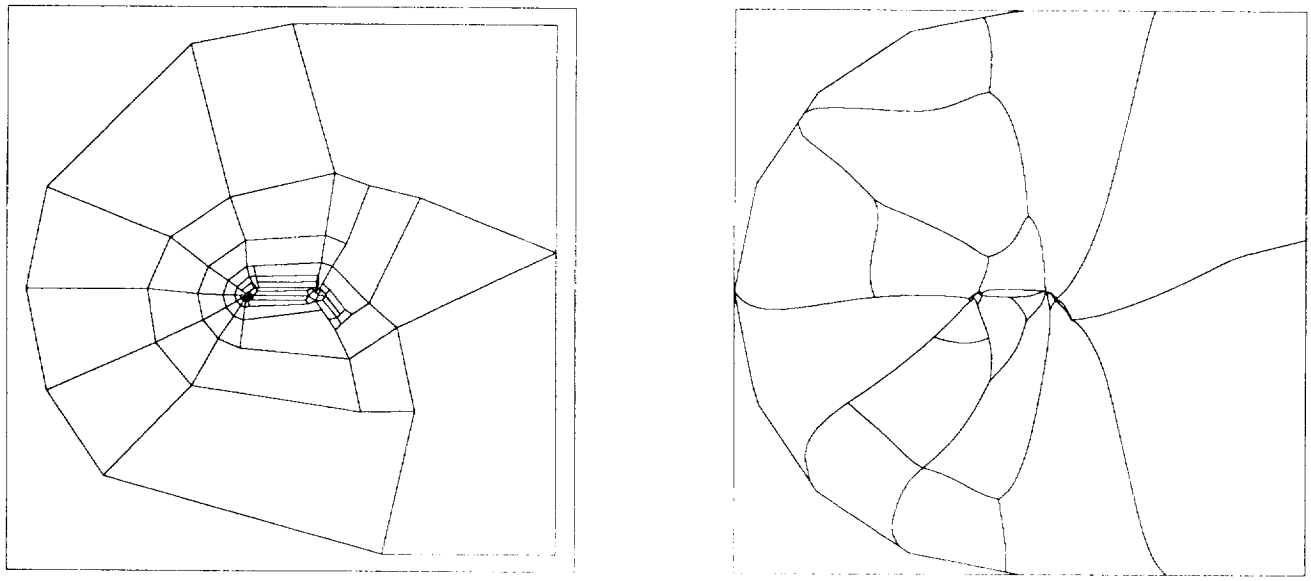


Figure 17: Multi-element airfoil: global block topology for method 1 (left) and method 2.

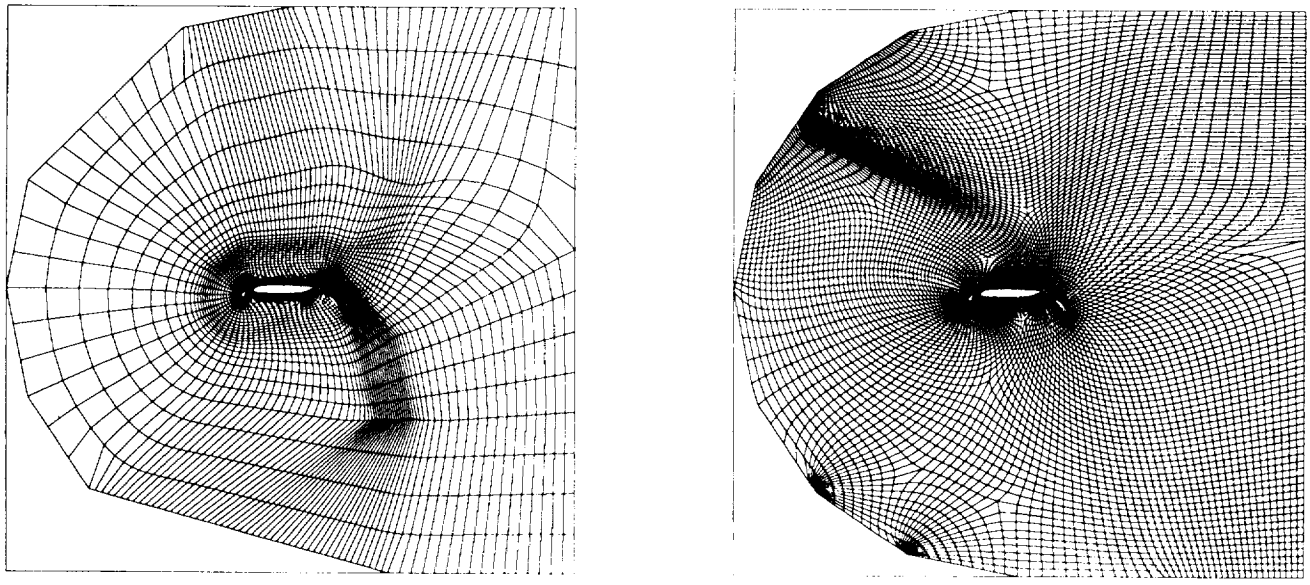


Figure 18: Multi-element airfoil: global grid for method (left) and method 2.

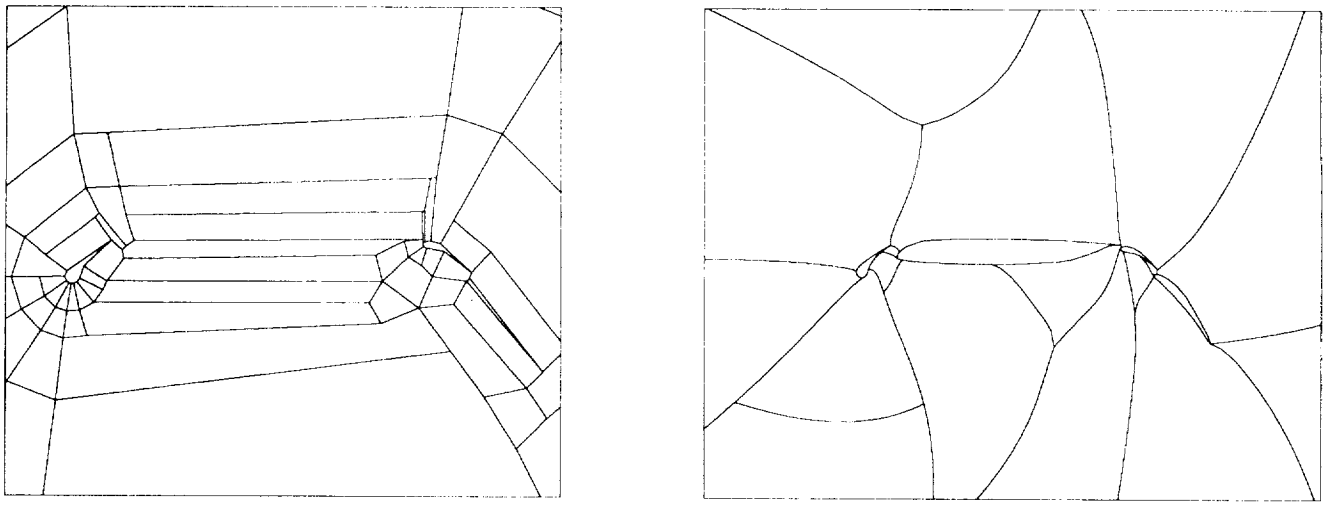


Figure 19: Multi-element airfoil: close-up of block topology for method 1 (left) and method 2.

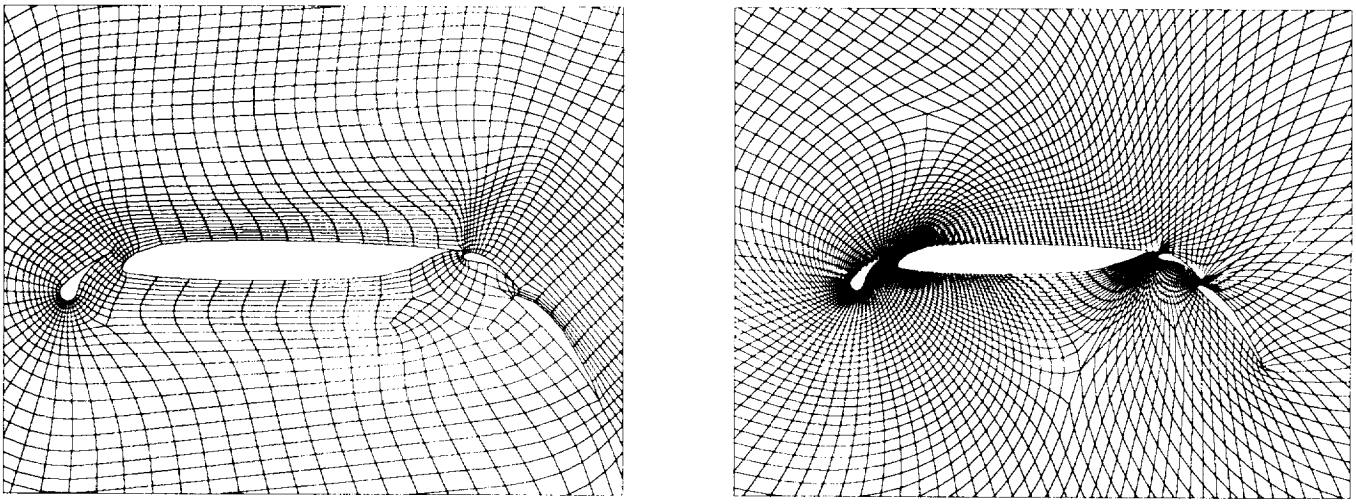


Figure 20: Multi-element airfoil: close-up of mesh for method 1 (left) and method 2.

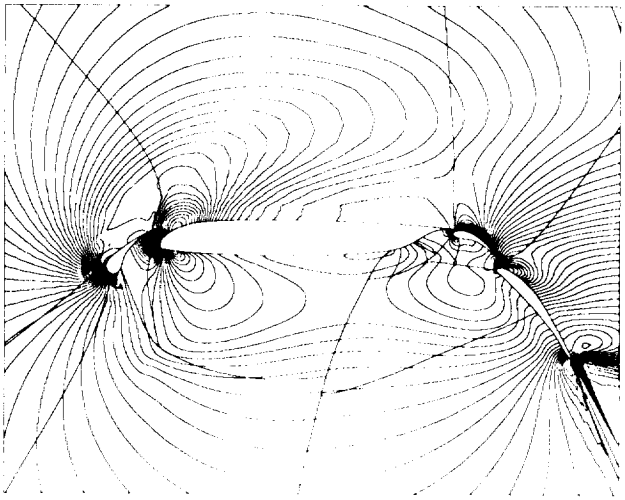


Figure 21: Multi-element airfoil: Mach number field for method 2.

N95- 28758

# INTELLIGENT AUTOMATED SURFACE GRID GENERATION

Ke-Thia Yao and Andrew Gelsey
Computer Science Department
Rutgers University
New Brunswick, New Jersey

**Abstract**

The goal of our research is to produce a flexible, general grid generator for automated use by other programs, such as numerical optimizers. The current trend in the gridding field is toward interactive gridding. Interactive gridding more readily taps into the spatial reasoning abilities of the human user through the use of a graphical interface with a mouse. However, a sometimes fruitful approach to generating new designs is to apply an optimizer with shape modification operators to improve an initial design. In order for this approach to be useful, the optimizer must be able to *automatically* grid and evaluate the candidate designs. This paper describes an intelligent gridder that is capable of analyzing the topology of the spatial domain and predicting approximate physical behaviors based on the geometry of the spatial domain to automatically generate grids for computational fluid dynamics simulators. Typically gridding programs are given a *partitioning* of the spatial domain to assist the gridder. Our gridder is capable of performing this partitioning. This enables the gridder to automatically grid spatial domains of wide range of configurations.

## 1 Introduction

Partial differential equation solvers require a grid, a discretization of the spatial regions of interest. In computational fluid dynamics, one type of the spatial regions of interest is the surface area the fluid contacts. The quality of the grid strongly affects the accuracy and the convergence properties of the resulting simulation. Generating a proper grid involves reasoning about the geometry of the regions of interest, the physics of the situation and the peculiarities of the numerical analysis code. To deal with the complexities of gridding, the current trend in the gridding field is toward interactive gridding [Remotique, Hart, & Stokes 1992, Kao & Su 1992]. Interactive gridding more readily taps into the

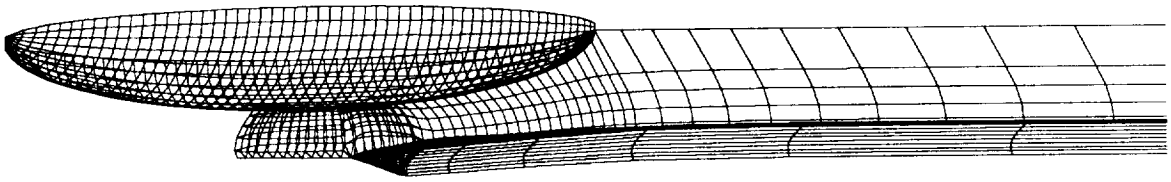


Figure 1: Yacht (consisting of three components hull, keel, and winglet) with wake sheets.

spatial reasoning abilities of the human user through the use of a graphical interface with a mouse. However, this approach is not acceptable for automated design systems, such as the Design Associate (DA) [Ellman, Keane, & Schwabacher 1992] for racing yachts. In the process of designing a yacht, the DA must repeatedly evaluate candidate yacht designs. A large number of these evaluations are required, so the capability to automatically evaluate the performance of a candidate yacht design without human intervention is crucial for the success of the DA.

We are working in the physical domain of fluid dynamics, in particular potential flows modeled by Laplace's partial differential equation. The potential flow solver we use is PMARC, a product of NASA Ames Research Center. The input PMARC requires is a panelization — a discretization of an object's wetted surface as a grid of *surface patches*, where each surface patch is an array of approximately planar quadrilateral panels. This array of panels is represented in PMARC as a matrix of corner points. See Figure 1 for a grid of a yacht automatically generated for PMARC by our gridding program.

The yacht in Figure 1 consists of three input components: an ellipsoid hull, the *Star & Stripes* keel, and the *Star & Stripes* winglet.[1] The wake sheets attached to the rear of the yacht are the vortices shed by the yacht. Discussions on how to attach wakes and how to determine the shape of the wakes are beyond the scope of this paper. The *Star & Stripes* winglet attached to the bottom of keel is considered a major innovation in the field of racing yachts, and the success of the *Star & Stripes* was in part due to its winglet. Current automated gridding programs should be but are not able to handle this kind of innovative *topological* change in design without human assistance.

The input to the gridder is expressed in a language we have developed called Boundary Surface Representation (BSR). Figure 2 graphically depicts the BSR input for this yacht example. We shall use this yacht example throughout this paper. Both BSR and the input will be discussed in much more detail later. For now we'll point out that BSR input consists of two major parts: *geometrical* and *topological*. The geometrical part represents the detailed features of the yacht, which are the three input surface mappings (*shape*) in the figure. The *topological* part contains information on the adjacency of the input surfaces.

[1] The *Star & Stripes* is the yacht that won the 1987 America's Cup Competition.

The adjacency information is represented by dotted lines in the figure.

# 2 Why is automated gridding hard?

## 2.1 Steps to gridding

We divide gridding into three steps. The first step is to *partition* the input surface into griddable *surface patches*. That is, this step finds the appropriate boundary lines (or *partitioning lines*) for the surface patches. As we'll see this step is often the most difficult, because it involves significant physical and geometrical reasoning.

Step two, for each surface patch, *reparametrize* it by defining two families of approximately orthogonal grid lines. A formal definition will be given later when BSR is defined. But, intuitively suppose a surface patch is laying on the $xy$-plane, then $\{x = constant, y = constant\}$ is one possible parametrization, and $\{x + y = constant, x - y = constant\}$ is another.

The last step is to determine how many grid lines to lay down on each of the surface patches, and in particular where to lay them down. This step corresponds to picking the constants to instantiate the equations in step two. The intersections of these grid lines form corner points of the array of panels, which is the input to PMARC. This step we shall call the grid line *distribution* step. The distribution of grid lines can make grids with the same reparametrization look different and may make the numerical simulator behave differently. For example, using the *equal-distance* distribution scheme, $x = i/10$, where $i = 0, \ldots, 10$, may make the numerical simulator converge slower than a *cosine* distribution scheme, $x = (1 - \cos \pi i/10)/2$, where $i = 0, \ldots, 10$.

## 2.2 Evaluation criteria

Gridding as defined by the three steps above is unconstrained. The ultimate test for a grid is to check how sound the resulting simulation is, and how well it resolves the physical features of the domain. Short of feeding the grid to a simulator, there are ways of checking the goodness of a grid.

Through our discussion with hydrodynamicists we have formulated a list of grid evaluation criteria and constraints. On the basis of the geometric properties of the grid, these evaluation criteria attempt to predict the soundness of PMARC's output. We divide this list into four levels, ranging from constraints that absolutely must be satisfied to heuristic advice based on experiences of our experts.

1. Simple connectedness constraint: surface patches must be simply connected, i.e., no holes.
2. Coverage constraint: patches must not overlap or leave gaps.
3. Planarity criterion: panels must be approximately planar.

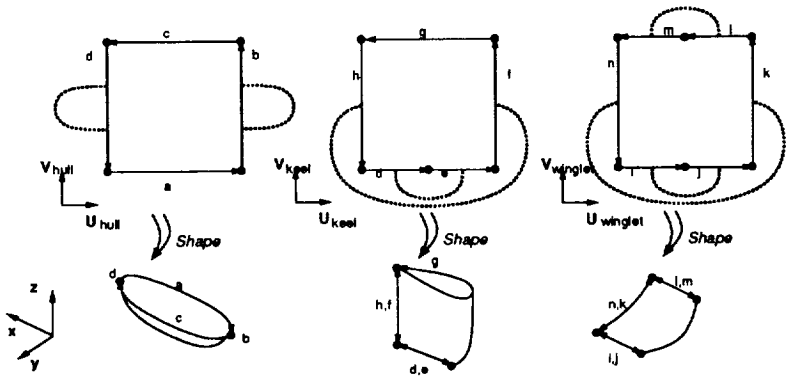



Figure 2: BSR input

4. Heuristic criteria:
   - following streamlines: grid lines should follow the streamlines of the fluid flowing over the body.
   - orthogonality: grid lines should intersect at right angles.
   - expansion ratio: the area of the adjacent panels should not increase by more than a fixed ratio.

## 2.3 Difficulties of partitioning

Much work has been done on the problem of automated gridding [Thompson, Warsi, & Mastin 1985], and many gridding programs have been developed. However, most of these efforts concentrate on developing new methods of reparametrization and new distribution schemes. The choices of which reparametrization method and which distribution scheme to use are usually left to the human expert.

Most of the programs rely exclusively on the human expert to do the partitioning. He is expected to do the partitioning by either writing batch commands, or more recently by using an interactive graphical interface. In either case, the partitions created only apply to the one particular problem at hand. More recently, [Schuster 1992] has been trying to revive batch mode gridding by writing more general batch commands. However, his program is only able to grid a small, fixed set of airplane topologies.

One of the fundamental problems with the current gridding programs is that they do not make use of topology. All the topological information has been distilled away by either having the user provide the partitions or by fixing the possible topologies. The programs can only work on individual surface patches. Another problem is that programs have neither the knowledge of physics nor the knowledge of numerical analysis needed to generate grids that will lead to good simulations.

One manifestation of the lack of physical knowledge is as follows. A closer examination of the surface area near where the hull and keel meet reveals that the keel actually protrudes into the hull, and the hull has an extra surface area where the keel is. Surfaces given to the gridding program often contain *fictional surface areas*, areas that should not be gridded. Fictional surface areas are useful because they allow the hull and keel to be modified independently while still remaining in contact. However, an automated gridding program must be able to distinguish between the real and fictional areas in order to satisfy the *coverage* constraint.

Recall that PMARC represents each patch by a matrix of corner points. This type of representation does not allow for holes in patches, i.e., the patches must be *simply-connected.* If the gridding program has knowledge of the underlying numerical analysis program, it would realize that once it removes the fictional surface area from the hull, it must break the hull in half to "cut" out the hole. This cut can be performed in limitless ways, but how it is done affects how easily the reparametrization and distribution steps can be performed to satisfy the evaluation criteria.

In the following sections we present a geometric language, Boundary Surface Representation (BSR), which is capable of representing geometrical information, topological information as well as associating attributes of the physical domain to the geometry. Also, we present a principled method of solving the partitioning, reparametrization, and distribution problems based on reasoning about physics of the flow domain. We call this method streamline-based gridding.

# 3 Boundary Surface Representation(BSR)

Surfaces are basically two dimensional objects that reside in three dimensional space. So they are naturally represented parametrically as a mapping from parametric space, $(u, v) = ([0, \ldots, 1], [0, \ldots, 1])$, to 3D Cartesian space, $(x, y, z)$. Our gridding system provides a mapping facility to represent this *shape* mapping, see Figure 2. No assumption is made about what mathematical form the mappings may take. Each mapping is treated as a "black box". The advantage of using a black box representation is that it provides greater flexibility by hiding the implementation details from the gridder. In our example, the hull is defined using algebraic formulae, and the keel and winglet are defined using B-spline surfaces.

This mapping facility is not limited to defining shapes. Other geometric and physical values may also be defined. For example, the outward normals of a surface may be defined as a *normal* mapping from the parametric space, $(u, v)$, to 3D vector space. Then in turn based on the *shape* and *normal* mappings, our gridding program can approximate the stream vectors on the surfaces as a *flow* mapping by projecting the *free stream vector*, $(1, 0, 0)$, onto the surface. The free stream vector is the direction the water would flow if the yacht were not

present.

Notice the boundaries of each surface are represented explicitly by directed edges, *arcs*. The arcs in turn are bounded by nodes. Explicit representation of the boundary is useful in that it allows for implicit representation of surfaces. That is, a closed sequence of arcs in parametric space can be used to denote the portion of the surface it encloses. The program adopts the *counter-clockwise rule*. A counter-clockwise, closed sequence of arcs denotes the area bound by the arcs. A clockwise, closed sequence of arcs denotes the area outside of the arcs. This implies the area on the "left-hand side" of an arc is "inside," and area on the "right-hand side" is "outside."

Arcs are also useful in expressing topological information. In our notation two arcs are connected by a dotted line if they are the same line when mapped using *shape* into $xyz$-space, even though they are distinct in parametric space. For example, in Figure 2 the keel parametric arcs $h$ ($u_{keel} = 0$) and $f$ ($u_{keel} = 1$) are connected by a dotted line, because both of these arcs map to the trailing edge of the keel. Thus in $xyz$-space it is possible to travel just in the direction of increasing $u_{keel}$ and end up at your starting point. This dotted line together with the dotted line connecting arcs $d$ ($u_{keel} = [0, \ldots, 0.5]$) and $e$ ($u_{keel} = [0.5, \ldots, 1]$) implies the topology of the keel is similar to that of a cylinder with one end closed or a "cup."

Notice that the hull parametric arcs $b$ ($u_{hull} = 1$) and $d$ ($u_{hull} = 0$) are connected to themselves. This is used to show that arcs $b$ and $d$ are degenerate, i.e., they each map to one point in $xyz$-space. The arc $d$ maps into the trailing point of the hull; the arc $b$ maps into the leading point of the hull.

Also, notice each of the arcs on the winglet is connected to some other arc. This means that in $xyz$-space the winglet surface does not have any boundaries. Of the three components the winglet is the only one that actually encloses some finite volume in $xyz$-space.

BSR provides a set of surface patch manipulation operations, such as intersection of surfaces, and division of patches into sub-patches. Figure 4 depicts the patches after the partitioning step. Reparametrization and distribution operations also are supported, see Figure 5. Now, we can formally define reparametrization as a mapping from a unit square, defined in a new parametric space, say $(s, t)$, to a surface patch in $(u, v)$ parametric space.

# 4 Streamline-based reasoning

The solution to Laplace's equation depends neither on the current state of the flow nor on time, so the geometry of the object determines the solution. Since streamlines are key characteristics of the solution, analyzing how streamlines interact with geometry provides key insights to qualitative behaviors of Laplace's equation. These insights enable us to determine the topology of streamlines. In turn this topology provides natural boundaries for patches in grids.

The most immediate reasoning problem we encounter in streamline-based reasoning is how to get the initial set of streamlines, since we have not yet run PMARC to generate the solution from which streamlines are extracted. We have experimented with various methods of predicting the streamlines *a priori*. However, we have found the simple projection of the *free stream* vector onto the body surface to be a good approximation of the true streamlines. This the *flow* mapping defined earlier.

## 4.1 Object classification

Analyzing the pattern of streamlines on the surface of different objects, we define two object classes. This first is the *source/sink node* class. Streamlines on objects from this class all originate from one point on the surface, the source node, and all flow to and terminate at another point on the surface, the sink node. Spheres, ellipsoids and other simple bodies of revolution are objects of this class. These objects have axial-symmetry, so there can only be one source node and one sink node.

The second is the *source/sink line* class. This class is like the previous class, except that the streamlines appear to originate and terminate at lines instead of nodes. For instance, the leading edge of a keel is *source line*, and the trailing edge is *Sink line*. All the streamlines flow from the leading edge to the trailing edge. Any wing shaped object belongs to this class.

Using only these two object classes, one can already construct complex, geometric objects, such as the yacht in this paper. The yacht consists of a source/sink node object (hull), and two source/sink line objects (keel and winglet). New classes can always be defined as the need arises.

## 4.2 Application to gridding

Based on the *following-streamline* heuristic for gridding, it is reasonable to grid a source/sink node object as a single surface patch, since all the streamlines are flowing in one direction, from the source node to the sink node. A source/sink line object should be gridded as two surface patches with the source line and sink line acting as partitioning lines. Although the streamlines still flow from the source line to the sink line, the streamlines take two different routes. For example, one set of streamlines flows to the sink from the right side of the keel ($u > 0.5$), and the other set flows from the left side ($u < 0.5$). The source/sink lines separate these two flow regions.

Streamlines are also useful in reparametrization. Streamlines can be defined as one family of grid lines. Lines orthogonal to the the streamlines can be defined as the other family. For example, on a sphere these two families correspond to the two spherical coordinate directions, $\theta$ and $\phi$, where $x = \cos\theta, y = \sin\phi\cos\theta, z = \sin\phi\sin\theta$. Streamlines have constant $\theta$ and the orthogonal lines have constant $\phi$.

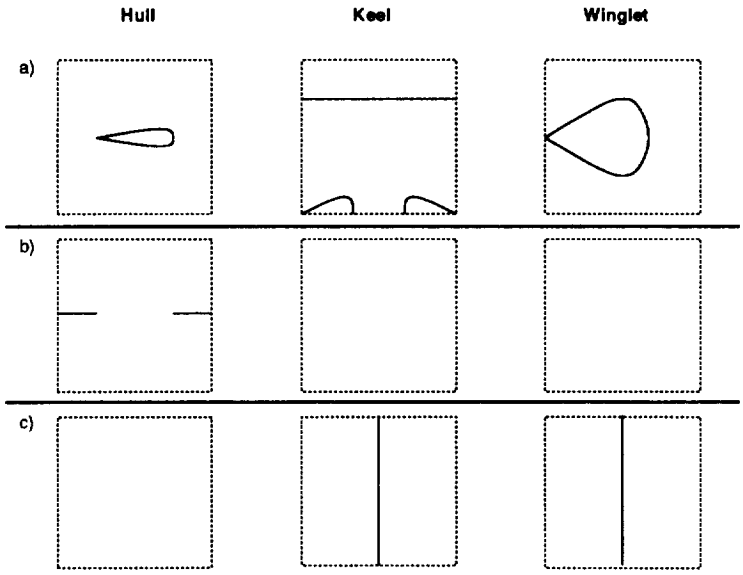



Figure 3: Partition lines: a) intersection lines, b) streamlines to "cut" holes out, and c) source/sink lines.

The sources and sinks provide guidelines on how to distribute the grid lines. The key to distributing grid lines is to highlight the physical features of the domain. That is, put more grid lines in regions where interesting physical changes occur. In the flow domain, the most interesting change is the change in direction and velocity of the flow. This change typically occurs most dramatically around the sources and sinks. So, the grid lines should be distributed more densely around them.

The above discussion deals with idealized objects. In the yacht example, there is a keel attached to the hull, and a winglet attached to the keel. The following sections show how to deal with the topological changes in these idealized objects by going through the three gridding steps in more detail.

# 5 Partitioning

We break the partitioning step into three sub-steps: 1) Determine the surface partitioning lines, 2) Partition surfaces into surface patches, and 3) Determine real surfaces patches.

## 5.1 partitioning lines

The partitioning lines that we use can be divided into three categories: surface intersection lines, streamlines, and source/sink lines. Intersection lines provide

the boundary between real and fictional surface areas, so they must be present. See Figure 3a for examples.

Notice that the hull-keel intersection line introduces a hole on the hull surface. This hole needs to be cut out, because of the *simply-connected* constraint. Using streamline-based reasoning, the logical way to "cut" out the hole is by cutting along streamlines. We search for a leading point and a trailing point along the intersection. From the leading point we trace a streamline *backward* along the hull surface. From the trailing point we trace a streamline *forward* along the hull surface. These two streamlines are shown in Figure 3b.

Source and sink lines are definitely needed, but all the sink lines turn out to be redundant. The source lines are shown in Figure 3c. Notice that source/sink nodes in $xyz$-space may become source/sink lines in $uv$ parametric space, as in the hull.

## 5.2 partition the surface patches

We shall not go into detail on how BSR accomplishes the actual partitioning. Basically BSR 1) gathers all the partition lines of a particular surface, 2) intersects the partition lines with each other and with the boundary lines of the surface, 3) breaks all the lines at intersections, 4) forms a wire frame from the broken lines, and 5) forms the surface patches based on the wire frame. The surface patches after partitioning are shown in Figure 4. BSR updates the topological information after the partitioning process. The shaded surface patches are fictional and will not be gridded.

## 5.3 determine the real surface patches

Real and fictional surface patches can be distinguished by reasoning using the outward *normal* mappings, the *counter-clockwise* rule, and intersection lines. For example, the surface patch $Keel1$, Figure 5, has intersection lines in common with the hull surface (arc 6) and the winglet surface (arcs 2 and 3). The hull outward *normals* along the hull-keel intersection generally point in the negative $z$-direction. The inward direction of arc 6 as defined by the counter-clockwise rule is in the negative $V_{keel}$ direction, which corresponds to the negative $z$-direction in $xyz$-space. This implies that $Keel1$ is outside of the hull. Similar reasoning using arcs 2 and 3 shows $Keel1$ is outside of the winglet. Since $Keel1$ is on the outside of all its neighbor surfaces, $Keel1$ is a real surface patch. If a surface patch is on the inside of one or more of its neighbors, then it is a fictional patch.

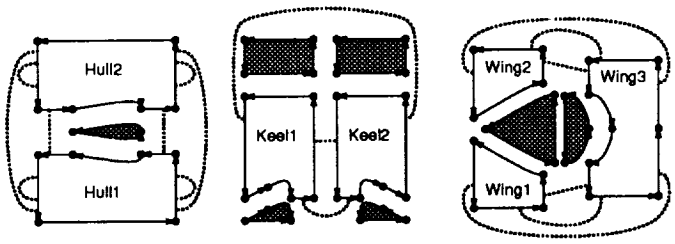



Figure 4: Surface patches after partitioning. Dotted lines across *uv*-space are not drawn to reduce clutter. Omitted dotted lines would show Keel1 connected to Hull1, Wing1 and Wing3, and would show Keel2 connected to Hull2, Wing2 and Wing3.

# 6 Reparametrization

Our gridding program is able to invoke several different reparametrization methods, including *transfinite interpolation*, *elliptic* methods, and *variational* methods, [Knupp & Steinberg 1993]. These methods map a unit square on to a quadrilateral. In the case of transfinite interpolation, the mapping is done by interpolating from opposite edges of that quadrilateral. All these method requires the surface patch to be reparametrized to have exactly four sides. But, surface patches tend to have more than four boundary edges, and sometimes less than four. In order to use these methods, we describe a heuristic, streamlined-based method of grouping the boundary arcs of the surface patches into four groups. See Figure 5.

We can classify each arc as either parallel, anti-parallel, or orthogonal with respect to the streamlines. For example, the patch *Keel*1 is bounded by six arcs. Arc 1 is a sink line. Arc 5 is a source line. So, by definition they are orthogonal to the streamlines. Arc 4 is a boundary arc from the original input surface. Arcs 2, 3 and 6 are intersection lines. These four arcs are neither completely parallel nor completely orthogonal to the streamlines. But by sampling different segments of these arcs, we can approximately classify arcs 2 and 4 as parallel, 6 as anti-parallel, and arc 3 as orthogonal. So, six groups are formed, $[(1),(2),(3),(4),(5),(6)]$. But, unlike the graphical depiction in Figure 5, arc 3 is very short when compared to its neighbors, arc 2 and 4. So, heuristically merging arc 3 with its neighbors, we get four groups, $[(1),(2,3,4),(5),(6)]$. We call this a *valid sequence* of groups, because it follows a [orthogonal, parallel, orthogonal, anti-parallel] pattern.

If there are not enough arcs to form four groups, then additional degenerate, point arcs may be introduced to construct a valid sequence of groups. For example, in the case of three arcs with classification [orthogonal, orthogonal, anti-parallel], then a parallel, point arc may be add between the two orthogonal

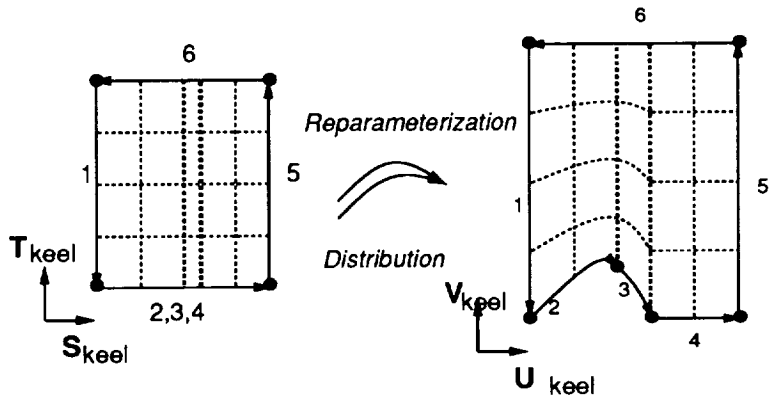



Figure 5: Reparametrization and Distribution

arcs.

Our grouping method works well, because the boundary arcs of the surfaces patches tend to be partitioning lines: intersection lines, streamlines, and source/sink lines. Classification of streamlines and source/sink lines are straightforward. In practice intersection lines tend always to be parallel, because an orthogonal intersection line causes too much drag, and would not be used in properly designed streamlined bodies.

This reparametrization step may fail in two cases. The first case occurs when the patch is inherently non-rectangular (e.g., a L-shaped surface patch), and the heuristic method fails to group the boundary arcs into four groups. The second case occurs when the patch is highly non-convex, and the repamaterization method fails to converge or returns a folded grid. In both case the geometry of the surface patch is too complicated, and additional partitioning lines are needed.

# 7 Distribution

According to streamline-based reasoning, grid lines should be concentrated more densely around sources and sinks. Sources and sinks tend to be at the ends of the surface patches (in Figure 5 arc 1 and arc 5) in our streamline-based gridding method. So, complicated distribution schemes usually are not needed. We have experimented with *cosine* and *hyperbolic tangent* schemes, which distribute more grid lines at the ends and yet distribute them smoothly enough as not to violate the *expansion ratio* constraint. Both schemes work well, but if many grid lines are laid out, *cosine* tends to place grid lines too densely at the ends. This leads to numerical truncation error.

Beside resolving physical features, distribution must also resolve geometric

features. For example, one $S_{keel} = constant$ grid line must be laid out at the intersection of arc 2 and arc 3, and another one grid line at the intersection of arc 3 and arc 4. Grid lines that must be laid out are shown as heavy, dotted lines in Figure 5. The node at the intersection of arc 3 and arc 4 touches three surface patches, $Keel1$, $Keel2$, and $Wing3$. Not laying a grid line at that node would create a gap there so the three patches would not meet.

# 8 Computational Results

Our gridding algorithms have been implemented in a working program. Figure 6 shows the results of a convergence study in which our gridding program generated a series of grids for PMARC. In the convergence study we ran a series of simulations using grids with the same partitioning and reparametrization, but with increasingly denser grid lines. As the grid becomes denser and grid spacing decreases, output quantities computed by PMARC should converge to their correct values. The output quantity we are most interested in is effective draft, a measure of the efficiency of a sailing yacht's keel. Figure 7 shows how effective draft converges as grid spacing is reduced in our convergence study.

Other values in Figure 6 can also be used as checks on the soundness of the simulation. For example, the maximum $C_p$ (pressure coefficient) should approach 1 as the grid is refined, and the minimum $C_p$ should not become too negative, as very large negative values usually indicate flaws in the grid. [Gelsey 1992] discusses automated evaluation of simulation output quality.

| Panels | Draft | Drag | Lift | min $C_p$ | max $C_p$ |
|---|---|---|---|---|---|
| 286 | 2.8836 | 0.0951 | -0.2879 | -3.1832 | 0.6191 |
| 938 | 2.5869 | 0.1255 | -0.2532 | -4.6730 | 0.7941 |
| 3678 | 2.4655 | 0.1479 | -0.2418 | -4.1283 | 0.9010 |
| 14898 | 2.4327 | 0.1676 | -0.2139 | -4.9392 | 0.9631 |

Figure 6: Convergence study

# 9 Future Work

This work can be extended in various directions. One is to add feedback and local refinement capabilities to the gridder. The streamlines predicted by PMARC may be fed back into the gridder to improve the grid. Also, the gridder can be extended to detect and correct local flaws in the grid based on intermediate values, such as the coefficient of pressure. Another direction is to extend the gridder to other physical domains where PDE simulators are needed. We believe our methodology of identifying key physical features of the domain and of reasoning about how they interact with the geometry is quite general and extensible. For example, in the ingot casting problem of heat transfer the temperature profile

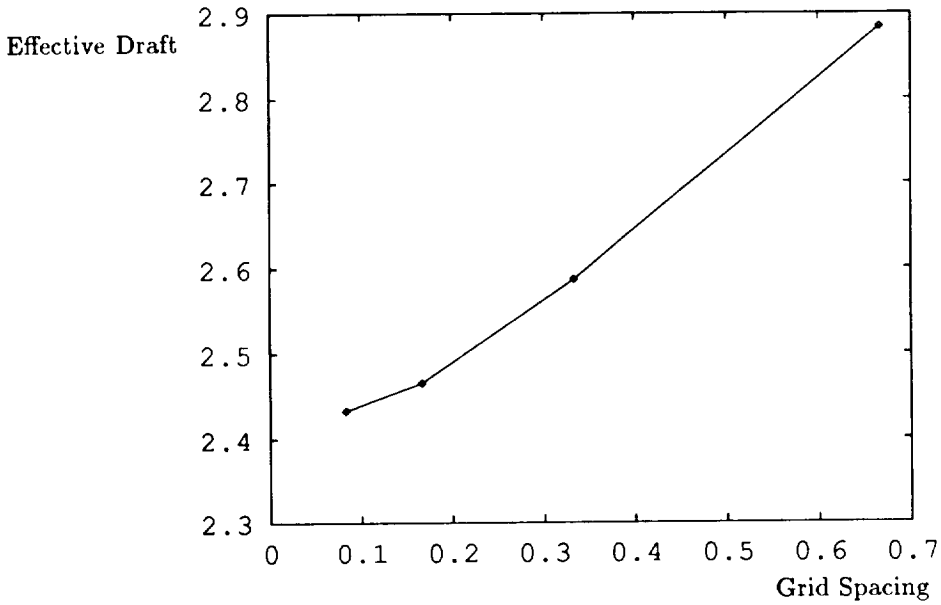



Figure 7: Convergence as spacing is reduced

seems to be the key feature [Ling, Steinberg, & Jaluria 1993]. Temperature profiles tend to change the fastest near sharp corners and in appendages (regions where the surface area to volume ratio is large). This suggests that isotherms should be useful as grid lines, and they should be distributed more densely near corners and appendages.

# 10 Related Work

Using streamlines is a natural idea. [Chung, Kuwahara, & Richmond 1993] defines a 2D finite-difference method based on streamline-coordinates, instead of Cartesian coordinates. [Chao & Liu 1991] applies streamline-based gridding to 2D flow problems consisting of a single patch. Many geometric modeling systems have been developed, such as Alpha1 by [Riesenfeld 1981] and SHAPES by [Sinha 1992]. [Requicha 1980] provides a good survey. Most of these systems are intended for modeling mechanical components, and provide little support for gridding, like representation of parametric space objects for reparametrization and distribution, and algorithms to manipulate these objects. Previous AI work in gridding includes [Dannenhoffer 1992], and [Santhanam *et al.* 1992]. In the 2D planar flow domain, Dannenhoffer's program is able to do partitioning by merging *templates* of previously-solved cases. So, the set of shapes it can handle is limited. Santhanam identifies several key parameters to modify and improve grids in 1D Euler domain.

# 11 Conclusion

Numerical simulation of partial differential equations is a powerful tool for engineering design. However, human expertise and spatial reasoning abilities are needed in order to form the spatial grids which PDE solvers require as input. We have developed a geometric modeling language, BSR, capable of expressing geometrical, topological, and physical aspects of the gridding problem, and we have used BSR as a basis for an intelligent automated system for generating the grids required for numerical simulation. The grid generation process involves analyzing the topology of the spatial domain, predicting and classifying the interactions of physics and geometry, and reasoning about the peculiarities of the numerical simulator.

# 12 Acknowledgments

The research was done in consultation with Rutgers Computer Science Dept. faculty member Gerard Richter. We worked with hydrodynamicists Martin Fritts and Nils Salvesen of Science Applications International Corp., and John Letcher of Aero-Hydro Inc. Our research benefited significantly from interaction with the members of the Rutgers AI/Design group. This research was partially supported by NSF grant CCR-9209793, ARPA/NASA grant NAG2-645, and ARPA contract ARPA-DAST 63-93-C-0064.

# References

[Chao & Liu 1991] Chao, Y. C., and Liu, S. S. 1991. Streamline adaptive grid method for complex flow computation. *Numerical Heat Transfer, Part B* 20:145–168.

[Chung, Kuwahara, & Richmond 1993] Chung, S. G.; Kuwahara, K.; and Richmond, O. 1993. Streamline-coordinate finite-difference method for hot metal deformations. *Journal of Computational Physics* 108:1–7.

[Dannenhoffer 1992] Dannenhoffer, J. F. 1992. Automatic block-structured grid generation — progress and challenge. In Kant, E.; Keller, R.; and Steinberg, S., eds., *AAAI Fall Symposium Series: Intelligent Scientific Computation*, 28–32.

[Ellman, Keane, & Schwabacher 1992] Ellman, T.; Keane, J.; and Schwabacher, M. 1992. The Rutgers CAP Project Design Associate. Technical Report CAP-TR-7, Department of Computer Science, Rutgers University.

[Gelsey 1992] Gelsey, A. 1992. Modeling and simulation for automated yacht design. In *AAAI Fall Symposium on Design from Physical Principles*, 44–49.

[Gelsey 1994] Gelsey, A. 1994. Automated reasoning about machines. *Artificial Intelligence.* To appear.

[Kao & Su 1992] Kao, T. J., and Su, T. Y. 1992. An interactive multi-block grid generation system. In Smith, R. E., ed., *Software Systems for Surface Modeling and Grid Generation*, number 3143 in NASA Conference Publication, 333–345.

[Knupp & Steinberg 1993] Knupp, P. M., and Steinberg, S. 1993. *The fundamentals of grid generation.* CRC Press.

[Ling, Steinberg, & Jaluria 1993] Ling, S. R.; Steinberg, L.; and Jaluria, Y. 1993. MSG: A computer system for auotmated modeling of heat transfer. *AI EDAM* 7(4):287–300.

[Remotique, Hart, & Stokes 1992] Remotique, M. G.; Hart, E. T.; and Stokes, M. L. 1992. EAGLEView: A surface and grid generation program and its data management. In Smith, R. E., ed., *Software Systems for Surface Modeling and Grid Generation*, number 3143 in NASA Conference Publication, 243–251.

[Requicha 1980] Requicha, A. A. G. 1980. Representations for rigid solids: Theory, methods, and systems. *Computing Surveys* 12(4):437–464.

[Riesenfeld 1981] Riesenfeld, R. F. 1981. Using the oslo algorithm as a basis for CAD/CAM geometric modeling. In *Proc. NCGA National Conf.*, 345–356.

[Santhanam *et al.* 1992] Santhanam, T.; Browne, J.; Kallinderis, J.; and Miranker, D. 1992. A knowledge based approach to mesh optimization in CFD domain: 1D Euler code example. In Kant, E.; Keller, R.; and Steinberg, S., eds., *AAAI Fall Symposium Series: Intelligent Scientific Computation*, 115–118.

[Schuster 1992] Schuster, D. M. 1992. Batch mode grid generation: An endangered species? In Smith, R. E., ed., *Software Systems for Surface Modeling and Grid Generation*, number 3143 in NASA Conference Publication, 487–500.

[Sinha 1992] Sinha, P. 1992. Mixed dimensional objects in geometric modeling. In *New Technologies in CAD/CAM.*

[Thompson, Warsi, & Mastin 1985] Thompson, J. F.; Warsi, Z. U. A.; and Mastin, C. W. 1985. *Numerical grid generation : foundations and applications.* North-Holland, Amsterdam.

N95- 28759

# OPTIMAL DOMAIN DECOMPOSITION STRATEGIES

Yonghyun Yoon*
Department of Aerospace Engineering
Korea Air Force Academy
Cheong-Ju, Korea

and

Bharat K. Soni**
NSF Engineering Research Center for
Computational Field Simulation
Mississippi State University
Mississippi State, Mississippi

## ABSTRACT

The primary interest of the authors is in the area of grid generation, in paticular, optimal domain decomposion about realistic configurations. A grid generation procedure with optimal blocking strategies has been developed to generate multi-block grids for a circular-to-rectangular transition duct. The focus of this study is the domain decomposition which optimizes solution algorithm/block compatibility based on geometrical complexities as well as the physical characteristics of flow field. The progress realized in this study is summarized in this paper.

## INTRODUCTION

Most solution algorithms for irregular configurations presently solve a discrete form of the fluid equations of motion. The discretized equations must be solved on a discretized computational region, that is, on the grid nodes. A discretization process of grid generation is to establish a relationship between the physical and computational domain, thus allow the solution algorithm to be performed in the transformed domain.

In principle it is possible to make a correspondence between any physical region and a single computational region. However, for general complicated three-dimensional configurations the resulting grid is likely to exhibit excessive skewness and coarseness. Despite the power and sophistication of present grid generation capabilities, it remains difficult to generate a reasonable, single-block grid about geometrically complex flow field configuration.

---

* Assistant Professor of Korea Air Force Academy
**Professor of Aerospace Engineering, Mississippi State University



PRECEDING PAGE BLANK NOT FILMED

A better approach with complicated physical boundaries is to decompose the physical domains, each bounded by six curved sides and each of which transforms to a rectangular block in the computational domain(ref.1). This domain decomposition has the merits of flow solver efficiency, grid smoothness and orthogonality, and flexibility of gridding for complex geometries by choosing suitable block topologies.

There are many approaches and philosophies which can be included under the domain decomposition strategies. Structured grid analysis of flow around complex geometries in widespread use is blocked decomposition, in which the solution domain is divided into regions with common internal boundaries. The grid lines at adjoining blocks can be set up to match in a variety of ways, with various levels of slope continuity. There is no theory which governs the way in which a flow field should be partitioned. A limited number of papers in the literature(ref.2-5) address basic flow-field-decomposition criteria and guidelines. Eiseman has developed an automatic block decomposition algorithm(ref.6). This algorithm is based on the geometrical complexities. However, no comprehensive, systematic studies have been done to determine the effect on the computed solution of using different blocks for the same geometry. Currently, the major bottleneck is in the design and implementation of the blocking plane(ref.7)

To remedy this problem, the focus of this study is the domain decomposition which optimizes algorithm/block compatibility based on geometrical complexities as well as the physical characteristics of the flow field.

Generally, the use of analytical shapes is not enough to satisfy the unusual geometrical requirement. Hence, sculptured curve/surface definitions such as Non-Uniform Rational B-Splines(NURBS), and quick elliptic grid refinement algorithms are developed. The application of these algorithms to grid adaption and domain decomposition is demonstrated. Grid generation associated with the circular-to-rect angular transition duct has been accomplished by applying these techniques. After careful consideration of the various alternatives, the structured muli-block approach must be chosen as the most suitable, from the point of view both grid generation and flow analysis. INS3D(ref.8), a three-dimensional multi-block incompressible Naver-Stokes code is used for this study.

## DOMAIN DECOMPOSION PROCEDURE

The work focuses on effects for optimizing domain decomposition strategies associated with circlar-to-rectangular transition duct. The transition duct is designed to connect a typical circular engine exhaust to a high aspect ratio rectangular nozzle. The application presented is of considerable engineering importance in internal fluid flow designs.

To take full advantage of the flexibility of multi-block structured techniques, one has to decide upon a suitable blocking topology to yield an optimal block arrangement for a given flow solver.

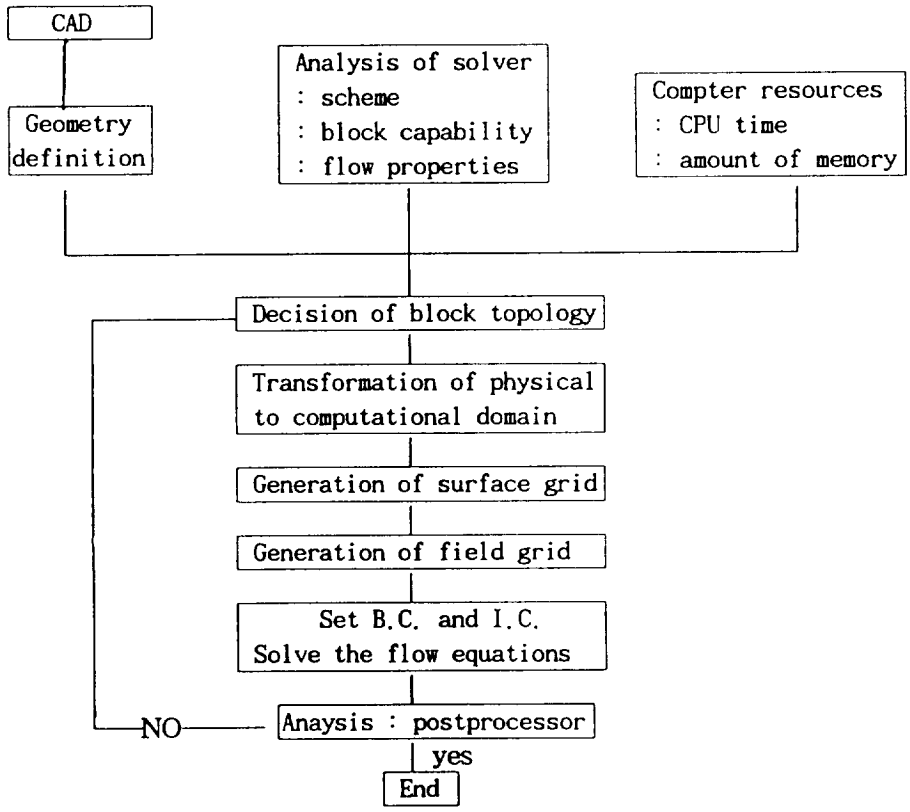



Figure 1. Basic procedure for domain decomposition

This step is an art, requiring knowledge-based techniques and trial-and-error. Typically, a domain decomposition strategy is performed and then surfaces and volume grids are created for each individual block. Figure 1 shows the basic procedure. The desired characteristics of multi-block grid are strongly dependent on the needs of the flow solver and computer.

## GEOMETRY DEFINITION FOR TRANSITION DUCT

The circle-to-rectangular transition duct can be designed by the equation of a superellipse.

$$\left(\frac{y}{a}\right)^{\eta} + \left(\frac{z}{a}\right)^{\eta} = 1 \qquad (1)$$

Where a and b are the semi-major and semi-minor axes, respectively, of the superellipse. The x axis is coincident with the streamwise direction and the y and z axes are parallel to the major and minor axes of the duct. The cross-sectional area at a given streamwise location is given by

$$A_{cs} = \frac{\Gamma(\frac{1}{\eta})^2}{\Gamma(\frac{1}{\eta})}(\frac{2}{\eta})(4ab) \qquad (2)$$

Where $\eta$ refers to the gamma fuction defined as

$$\Gamma(\eta) = \int_0^{\infty}(e^{-t}\, t^{\eta-1})dt \qquad (\eta>0) \qquad (3)$$

The design procedure for the transition duct is to specify the streamwise variation of the semi-major and semi-minor axes, and superellipse exponent($\eta$) defined by fifth order polynomial functions. Special cases of the superellipse include a circle( a=b, $\eta=\infty$), an ellipse( $a \neq b$, $\eta=\infty$).

Steady, incompressible, turbulent, swirl-free flow through a circle-to-rectangular transition duct has been studied experimentally (ref.9). For comparison, the same geometry has been simulated. Figure 2 shows the lower half of the duct. The transition duct has an inlet diameter of 20.43cm, a length-to-diameter ratio of 1, and an exit plane

aspect ratio of three. The transition region length-to-diameter ratio is 1.5. The cross-sectional area remains the same at the exit as at the inlet, but varies through the transition section to a maximum value approximately 15% above the inlet value.

## SURFACE REDISTRIBUTION AND REMAPPING

The principal requirement for generating a grid about a complex geometry is the ability to efficiently redistribute points on the surface while maintaining the integrity of the geometry. Because the regions where high gradients are expected in the flow solution require high density of grid points. The redistributed surface grid is accomplished by evaluating the NURBS surface at the respective parameter associated with the desired distribution space. The NURBS is used for the standard surface description. The convex hull, local support, and variation disminishing properties of B-spline functions contribute to the generation of the well-distributed smooth grid.

Let $r=[x_1(\xi,\eta),\ x_2(\xi,\eta),\ x_{3(\xi,\eta)}]$ denote the parametric representation of the surface with coordinate $(x_1,\ x_2,\ x_3)$ and parameter $(\xi,\ \eta)$. A control point form of the NURBS surface(in 3-D) is defined as a tensor product formula in 4-D(ref.10).

$$P(\xi,\ \eta)=\frac{\sum_{i=0}^{NI}\sum_{j=0}^{NJ}B_i^M(\xi)B_j^K(\eta)H_{ij}Q_{ij}}{\sum_{i=0}^{NI}\sum_{j=0}^{NJ}B_i^M(\xi)B_j^K(\eta)H_{ij}} \tag{4}$$

Where $B_i^M(\xi)$:the $M^{th}$ degree B-spline basis functions in $i$ direction.

$B_k^M(\eta)$:the $K^{th}$ degree B-spline basis functions in $j$ direction.

$H_{ij}$ : weight (positive real values)

$Q_{ij}$ : control points

The control points of the surface are determined using tensor product formula associated with both $\xi$ and $\eta$ parameters when a surface with a set of data point is given. It is called the inverse problem.

The parametric space associated with NURBS is transformed as the normalized arc length distribution mesh. The original surface which is expressed into non-NURBS form can be converted to NURBS representation. It is important to note that the redistributed surface grid

is obtained by evaluating the NURBS surface at the desired distribution point. However, the NURBS surface is smooth and has kept precisely the fidelity of the original surface. A redistributed surface resulting from the NURBS surface is demonstrated in Figure 3.

## ELLIPTIC REFINEMENT

The grid should exhibit the desirable qualities of smoothness and orthogonality especially near the solid surface. Some results of algebraic grid methods such as transfinite interpolation functions can not meet the requirement of the grid qualities on certain geometries. Thus, elliptic grid methods are used to make up the shortcoming of the algebraic methods(ref.11).

Derivation of the control function based on grid metrices begins with analyzing the elliptic grid equation. Define the physical and computational space with $r=(x_1, x_2, x_3)$ and $\Omega=(\xi^1, \xi^2, \xi^3)$, respectively. The covariant and contravariant vectors then appear as follows:

$a_i$ = covariant base vectors $r_{\xi^i}$ $(i=1, 2, 3)$

$a^i$ = contravariant base vectors $\nabla\xi^i$ $(i=1, 2, 3)$

$g_{ij} = a_i \cdot a_j = g^{ji}$ $(i=1, 2, 3)$, $(j=1, 2, 3)$

$g^{ij} = a^i \cdot a^j = g^{ji}$ $(i=1, 2, 3)$, $(j=1, 2, 3)$

$g = det|g_{ij}| = [\, a_1 \cdot (a_2 x a_3)]^2$

$(g_{ij})_{\xi^k} \equiv$ *derivative of* $g_{ij}$ *with respective to* $\xi^k$

$\equiv r_{\xi^i\xi^k} r_{\xi^j} + r_{\xi^i} r_{\xi^j\xi^k}$, $(i=1, 2, 3)$, $(j=1, 2, 3)$, $(k=1, 2, 3)$

The elliptic grid generation system used in this study(ref.12) is

$$\sum_{i=1}^{3}\sum_{j=1}^{3} g^{ij} r_{\xi^i\xi^j} + \sum_{k=1}^{3} g^{kk} P_k r_{\xi^k} = 0 \qquad (5)$$

The determinations of the three control functions for the general three-dimensional case can be summarized as follows. The three components of the elliptic grid generation equation(5) provide a set of three equations,

$$\sum_{k=1}^{3} g^{kk} (r_{\xi^k})_l P_k = -\sum_{i=1}^{3}\sum_{j=1}^{3} g^{ij} (r_{\xi^i\xi^j})_l, \quad l=1, 2, 3 \qquad (6)$$

that can be solved simultaneously at each point for the three control functions, $P_k$, $k=1, 2, 3$. The derivatives here are represented by central differences. The smoothness is established by replacing the control function at each point with the average of the four neighbors in the two curvilinear directions other than that of the function. Thus,

$$P_1(\xi^1, \xi^2, \xi^3) = \frac{1}{4}[P_1(\xi^1, \xi^2+1, \xi^3) + P_1(\xi^1, \xi^2-1, \xi^3) + P_1(\xi^1, \xi^2, \xi^3+1) + P_1(\xi^1, \xi^2, \xi^3-1), \quad (7)$$

with analogous equations for $P_2$ and $P_3$. No smoothing is done in the direction of the function because to do so would smooth the spacing distribution. The use of smoothed control functions evaluated from the algebraic grid produces a smooth grid that retains essentially the spacing of the algebraic grid(ref.13).

An application of these control functions results in a smooth-nearly orthogonal grid in fewer iterations of the elliptic solver. These control functions are applied in surface/volume grid refinement. It can be observed that the elliptic grid provides smoothness and near orthogonality in Figure 4.

## BLOCK INTERFACES

There are three distinct configurations along the axial direction which are a constant diameter circular entrance section, the transition section, and a rectangular extension section. Because of the drastic changes in the flow direction, a multi-block grid topology has been adapted. This provides a smooth discretization of the entire volume inside the transition duct.

Each block has its own curvilinear coordinate system irrespective of that in the adjacent blocks. In order to keep complete continuity of grid lines across the interfaces between blocks, the linkage among the various blocks can be set by interpolation or fixing an actual boundary. The grid points on an interface of one block are coincident in physical space with those on another interface of the same or another block. This facilitates the interface of block treatment without an application of interpolation. This philosophy allows the flow solver to be run on the multi-block grid system.

## APPLICATION

The two previously described methods, elliptic and algebraic, are merged appropriately to obtain various multi-block grids on the transition duct configuration. Moreover, by utilizating an approprite blending of these methods allows a grid generation capability ranging from cases with only a few blocks to cases with hundreds of blocks for various shapes.

To demonstrate the capability of the present procedure, several types of multi-block grids are designed such that the grid cell aspect ratio and orthogonality are maintained with a reasonable range for all the duct cross section. Figure 5-8 present several examples. For comparative purposes, all types of domain decomposition are implemented in the same flow solver code using equivalent boundary conditions. Even the flow conditions for the computational analysis are chosen to correspond to the experiments reported in Ref.8. Inlet conditions are as follows:

Reynolds number = 390,000(based on U and D)
Bulk velocity(U) = 29.95 m/s
Core velocity(Uc/U) = 1.083
Friction velocity(Utau/U) = 0.04063

## RESULTS AND DISCUSSION

The results of the flow calculation are shown in Figures 9-12. Pressure contours of the axial flow component are plotted in Figure 9 (a)-(b) which are on the x-y plane and x-z plane, respectively. The velocity contours on both planes are demonstrated in Figure 10.(a)-(b). The transition segment produces saddle-shaped pressure distributions in the y-z plane as shown in Figure 11. Within the boundary layer the velocity is reduced as shown in Figure 12, but the cross-stream pressure gradient imposed by the flow outside the boundary layer is not reduced. This can result in significant flow turning in the boundary layer and is refered to as skew-induced secondary flow or cross flow. The rate of cross flow production increases as the amount of streamline curvature increases(ref.13). Figure 13 shows the comparison of sidewall normalized static pressure between the experimental data and the computed solutions in the upper quarter of the duct only. The results of computational simulation and the measured data of Ref.9., show a very good agreement.

## CONCLUSIONS

Several blocking strategies have been considered in order to analyze the transition duct flow. Optimal domain decomposition is dependent on the method of flow solver and flow properties as well as geometry concerns. This domain decomposition gives us the following merits:

- A higher solution accuracy and faster convergence for the computational fluid dynamics solver.
- Savings in the CPU time and the amount of memory.
- Maximization of the grid quality and optimization of the grid distribution.

## REFERENCES


1. Thompson, J. F., : A General Three-Dimensional Elliptic Grid Generation System on A Composite Block Structure, Computer Methods in Applied Mechanics and Engineering, p.377, 1987.
2. Allwright, S. E., : Techniques in Multi-Block Domain Decomposition and Surface Grid Generation, Numerical Grid Generation in Computational Fluid Mechanics '88,pp.559-568, Pineridge Press Limited, Mumbles, Swansea, U.K., 1988.
3. Piperni, P. : Multiblock Grid Generation with CAD-Based Domain Decomposition Techniques,:Numerical Grid Generation in Computational Fluid Dynaimics and Related Fields, pp.109-121, Pineridge Press Limited, Mumbles, Swansea, U.K., 1994
4. Rossow, C. C. and Ronzheimer, A., : Multi-Block Grid Generation around Wing-Body-Engine-Pylon Configurations,Numerical Grid Generation in Computational Fluid Dynamics and Related Fields, North-Holland, New York, 1991.
5. Vogel, A. A., :A Knowledge-Based Approach to Automated Flow-Field Zoning for Computational Fluid Dynamics, NASA TM101072, 1989.
6. Eiseman, P. R. etc, : Applications of Multi-Block Grid Generations with Automatic Zoning,Numerical Grid Generation in Computational Fluid Dynaimics and Related Fields, Pineridge Press Limited, Mumbles, Swansea, pp.123-134. U.K., 1994

7. Dannehoffer, J. F., : A Block-Structuring Technique for General Geometries, AIAA Paper 91-0145, Jan. 1991.
8. Rogers, S. E., Kwak, D. C., and Chang, J. L. C., : INS3D- An Incompressible Navier-Stokes Code in Generalized Three-Dimensional Coordinates, NASA TM100012, 1987.
9. Davis, D. O., : Experimental Investigation of Turbulent Flow Through a Circle-To-Rectangular Transition Duct., NASA TM105210, 1992.
10. Yoon, Y. H., : Enhancements and Extensions of EAGLE Grid Generation System, Ph.D, Dissertation, Mississippi State University, 1991.
11. Thompson, J. F. Warsi, Z.U.A. and Mastin, C. W. : Numerical Grid Generation, Foundations and Applications, North-Holland , 1985.
12. Thompson, J. F. : Program EAGLE,Numerical Grid Generation System User's Manual,Vols.II and III,AFATL-TR-87-15,1988.
13. Reichert, B. A., Hingst, W. R., and Okiish, : Circular-to-Rectangular Transition Duct Flow without and with Inlet Swirl, Journal of Propulsion and Power. Vol.10, No.1, Jan-Feb. 1994.

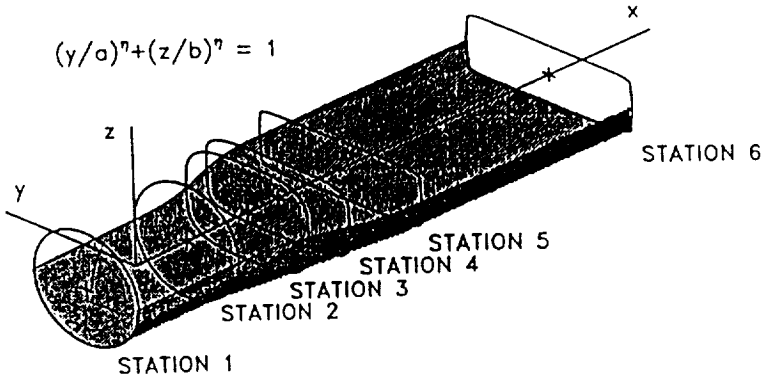



Figure 2. The lower half of the duct

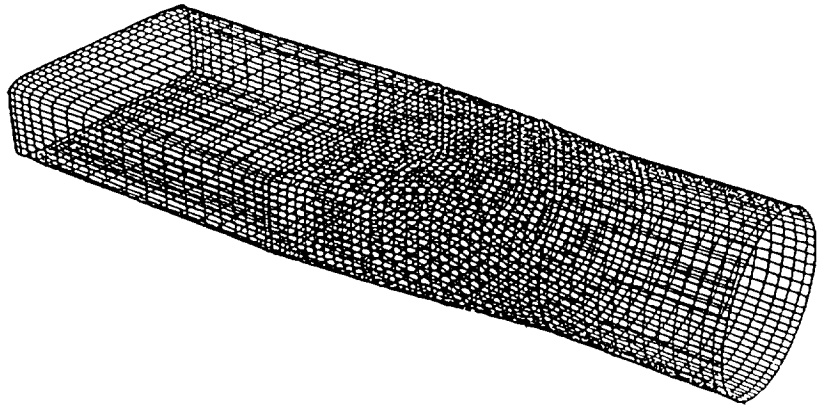


Figure 3. A NURBS surface

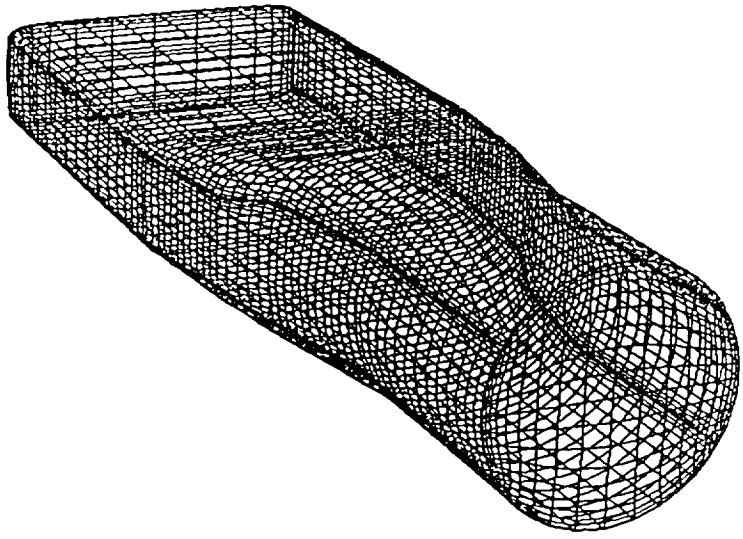


Figure 4. Elliptic grid refinement

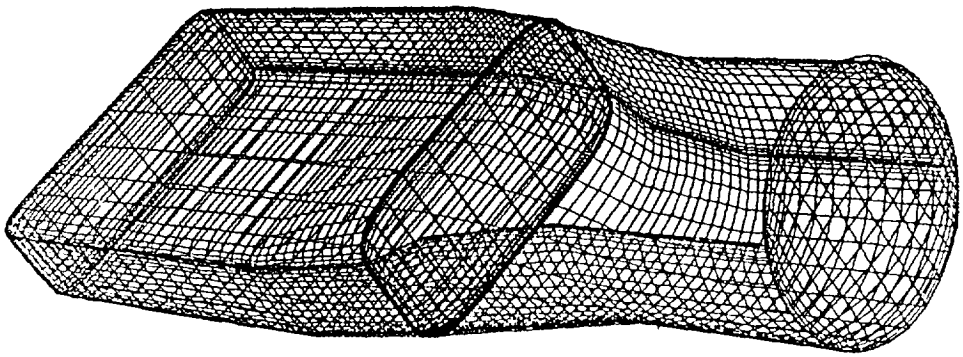


Figure 5. Two-block system along the flow direction

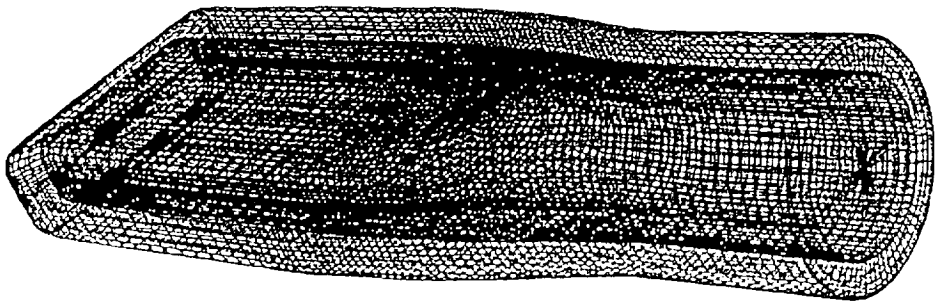


Figure 6. Two blocks, O-type grid system.

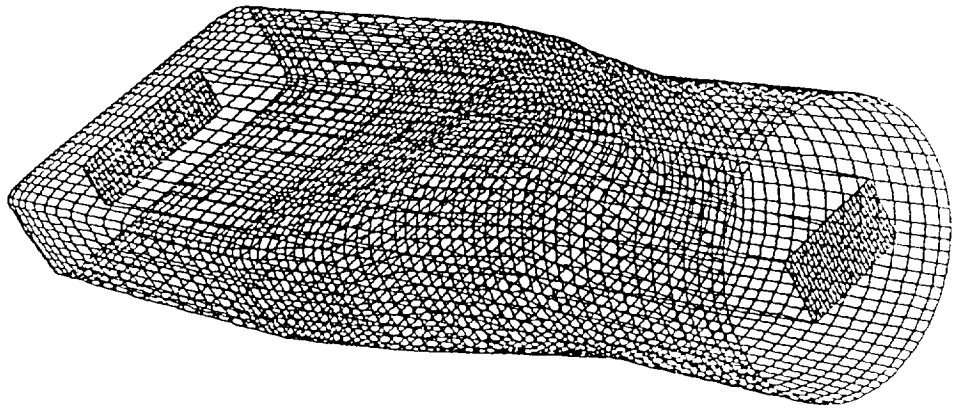


Figure 7. Two blocks, O-H type grid system.

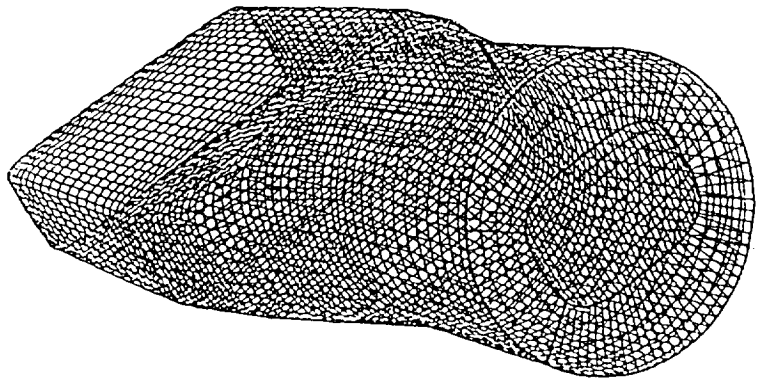
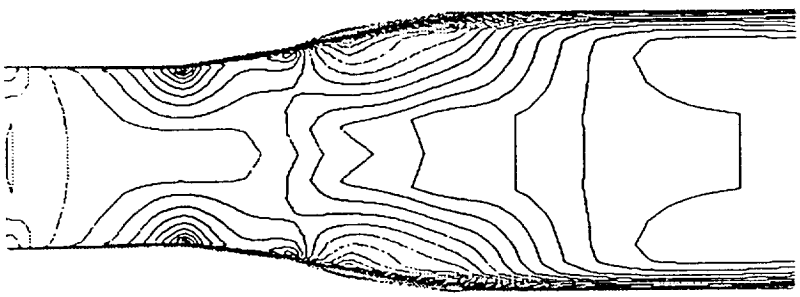


Figure 8. 5 blocks H-type grid system.

(a)



(b)

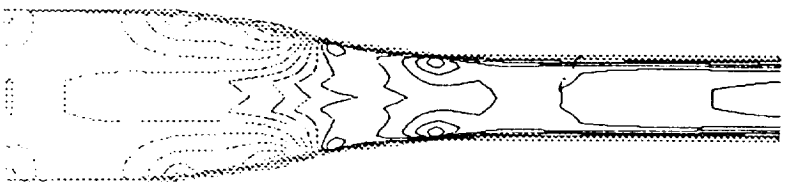
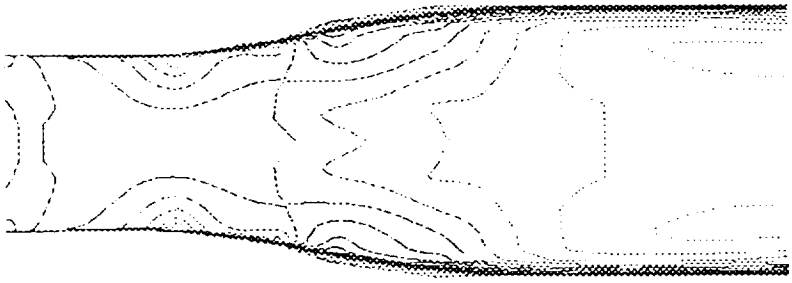


Figure 9. Pressure contours (a)x-y plane, (b)x-z plane.

(a)



(b)

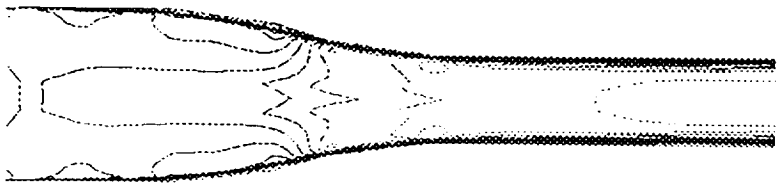


Figure 10. Velocity contours (a)x-y plane, (b)x-z plane.

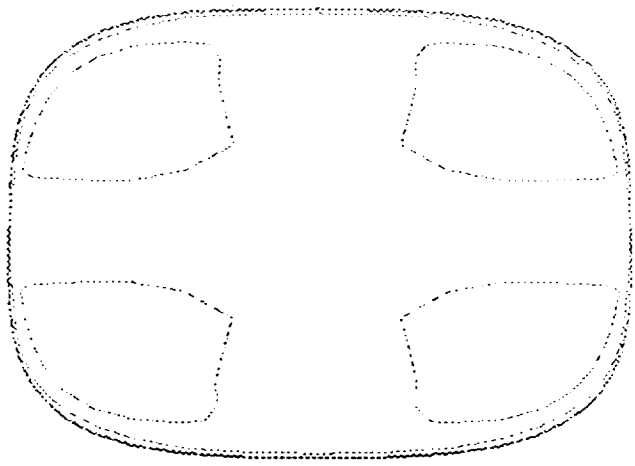


Figure 11. Vorticity contuors.

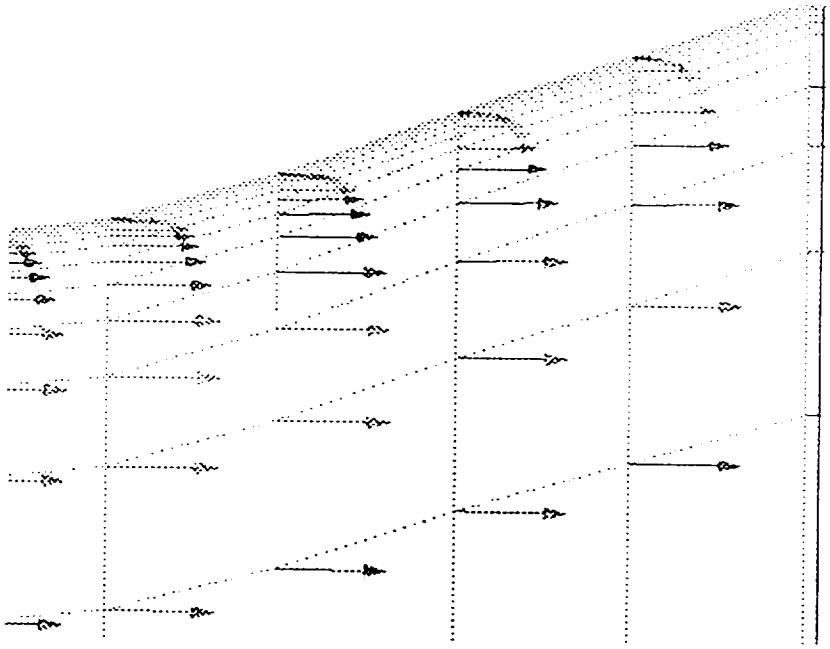


Figure 12. Boundary layer effects.

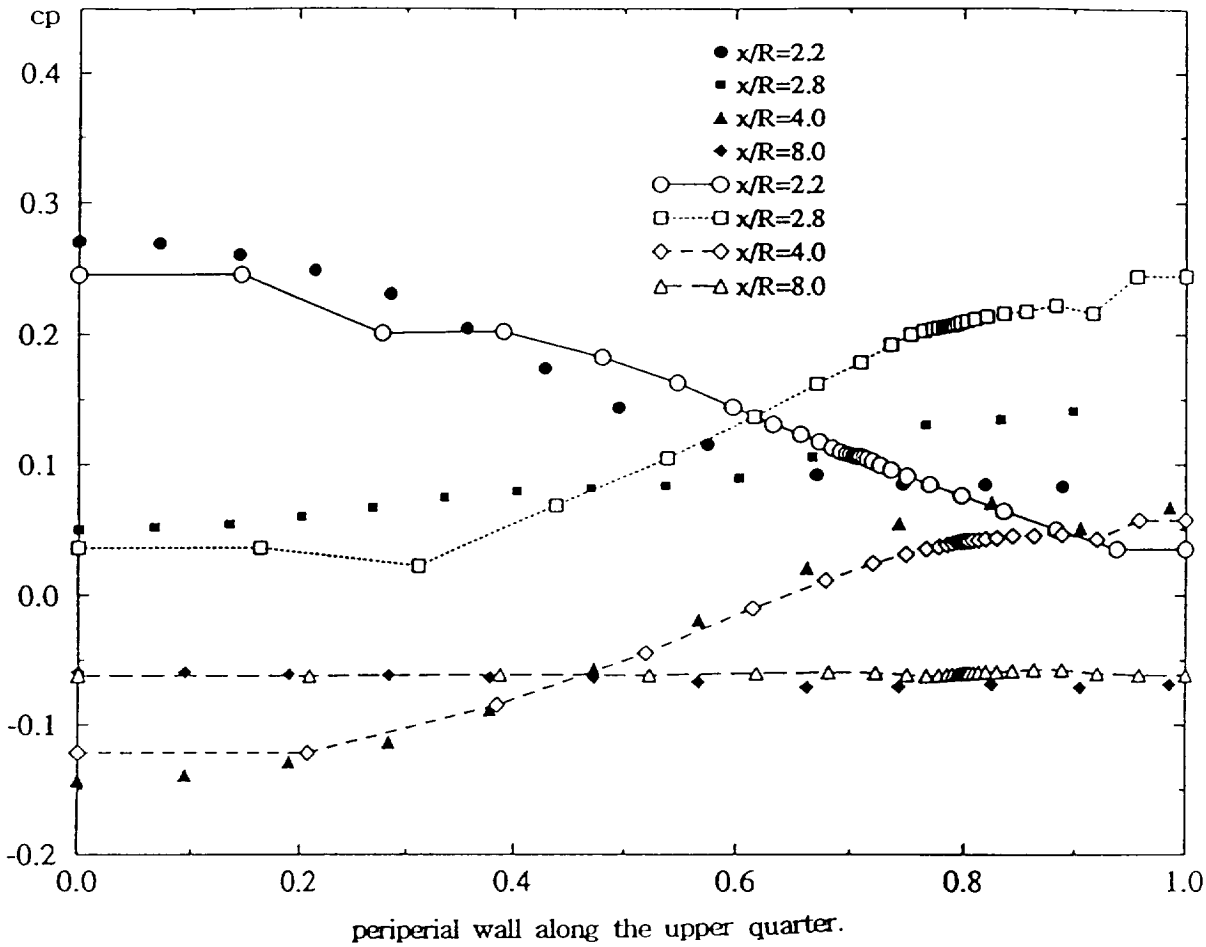



Figure 13. Comparison of peripheral wall pressure coefficient distribution between computed and measured data. (symbol: experiment, simbol-line:calculation)

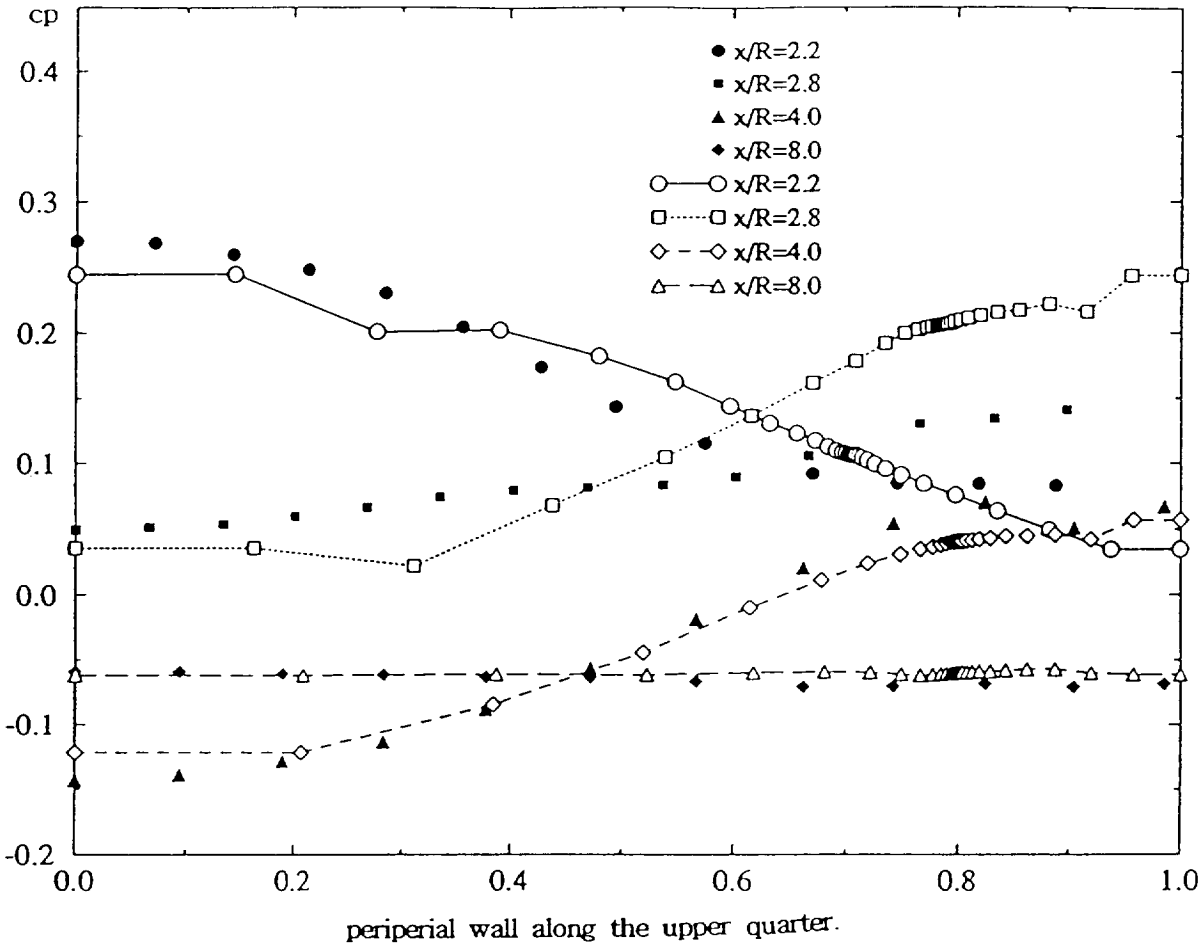



Figure 13. Comparison of peripheral wall pressure coefficient distribution between computed and measured data. (symbol: experiment, simbol-line:calculation)

# NEW METHODS/APPROACHES/ APPLICATIONS (1)

PRECEDING PAGE BLANK NOT FILMED

N95- 28760

# Elliptic Surface Grid Generation on Minimal and Parametrized Surfaces [1]

S.P. Spekreijse, G.H. Nijhuis, J.W. Boerstoel
National Aerospace Laboratory NLR
P.O. Box 90502, 1006 BM Amsterdam, The Netherlands

## Summary

- An elliptic grid generation method is presented which generates excellent boundary conforming grids in domains in 2D physical space. The method is based on the composition of an algebraic and elliptic transformation. The composite mapping obeys the familiar Poisson grid generation system with control functions specified by the algebraic transformation. New expressions are given for the control functions. Grid orthogonality at the boundary is achieved by modification of the algebraic transformation.
- It is shown that grid generation on a minimal surface in 3D physical space is in fact equivalent to grid generation in a domain in 2D physical space.
- A second elliptic grid generation method is presented which generates excellent boundary conforming grids on smooth surfaces. It is assumed that the surfaces are parametrized and that the parametrization is a smooth mapping from a unit square onto the surface. A generated surface grid only depends on the shape of the surface and is independent of the parametrization.
- Concerning surface modeling, it is shown that bicubic Hermite interpolation is an excellent method to generate a smooth surface which is passing through a given discrete set of control points. In contrast to bicubic spline interpolation, there is extra freedom to model the tangent and twist vectors such that spurious oscillations are prevented.

## 1 Introduction

A flow simulation system for the computation of flows about complete aircraft configurations including propulsion aircraft components has been developed at NLR. The system is known as the ENFLOW (Euler/Navier-Stokes FLOW) system [1]. Fig.4 shows the layout of the system and summarizes its use for CFD work.

Surface modeling of the original aerodynamic input configuration surfaces is done with the commercial ICEM-CFD software. The subdivision of a three dimensional flow domain into blocks is done with the graphical interactive domain modeler ENDOMO. The computation of structured grids in the interior of the blocks is done with the graphical interactive grid generator ENGRID. Given a multi-block grid, the flow solver ENSOLV computes the solution of the Euler and/or Navier-Stokes equations with respect to specified boundary conditions.

In this paper we focus on surface grid generation. Output of ICEM-CFD is a set of discrete

[1]This investigation was partly performed under contract 01105N with the Netherlands Agency for Aerospace Programs (NIVR).

PRECEDING PAGE BLANK NOT FILMED

surfaces. A discrete surface is a two-dimensional array of so-called control points. The complete set of discrete surfaces approximates the original aerodynamic input configuration surfaces.

A discrete surface must be interpolated during grid generation. For this purpose, bicubic Hermite interpolation is used to define a smooth surface which is passing through the set of control points without introducing spurious oscillations. A $C^1$ parametrization is constructed which maps a unit square onto the interpolated surface. Bicubic Hermite interpolation is discussed in Section 5.

Multi-block grid generation proceeds from the inside out, starting with the generation of grids in edges, followed by the grid generation in faces, and ending with the grid generation in blocks. For this reason, a surface grid generation method is needed to generate interior grids in surfaces with a prescribed boundary grid point distribution as Dirichlet boundary condition.

In Section 4 it is shown how elliptic surface grid generation can be used to generate an interior surface grid on a parametrized surface with a prescribed boundary grid point distribution. A generated surface grid is independent of the parametrization. Thus the interior surface grid will only depend on the shape of the surface and the prescribed boundary grid points.

For surfaces in the interior of a flow domain, often only the boundary shape is defined. For a surface with only a prescribed boundary shape and a prescribed boundary grid point distribution, it is possible to generate an interior surface grid on a minimal surface. The shape of the surface becomes a soap film bounded by the prescribed boundary of the surface. It appears that surface grid generation on minimal surfaces is in fact a straightforward extension of grid generation in 2D.

Grid generation in a domain in 2D is treated in Section 2, the extension to minimal surface grid generation is discussed in Section 3.

Concerning grid generation, the main emphasis lies on the derivation of the elliptic grid generation systems. The discretization and solution of the systems of partial differential equations is not discussed. We refer to [2] for details about the applied solution methods. Grid generation in 3D domains is also discussed in [2].

## 2 2D grid generation

Consider a simply connected bounded domain $\mathcal{D}$ in two dimensional space with Cartesian coordinates $\vec{x} = (x, y)^T$. Suppose that $\mathcal{D}$ is bounded by four edges $E_1, E_2, E_3, E_4$. Let $(E_1, E_2)$ and $(E_3, E_4)$ be the two pairs of opposite edges as shown in Fig.1.

Introduce the parameter space $\mathcal{P}$ as the unit square in a two dimensional space with Cartesian coordinates $\vec{s} = (s, t)^T$. Require that the parameters $s$ and $t$ obey:

- $s \equiv 0$ at edge $E_1$ and $s \equiv 1$ at edge $E_2$, (1)
- $s$ is the normalized arc length along edges $E_3$ and $E_4$, (2)
- $t \equiv 0$ at edge $E_3$ and $t \equiv 1$ at edge $E_4$, (3)
- $t$ is the normalized arc length along edges $E_1$ and $E_2$. (4)

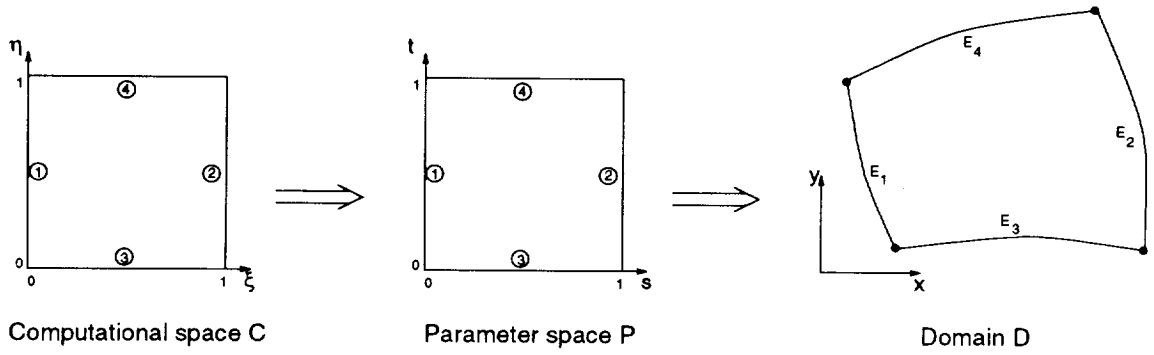



Figure 1: Transformation from computational $(\xi, \eta)$ space to a domain $\mathcal{D}$ in Cartesian $(x, y)$ space.

The mapping $\vec{s}: \partial\mathcal{D} \mapsto \partial\mathcal{P}$ is defined by these requirements. In the interior of $\mathcal{D}$ we require that $s$ and $t$ are harmonic functions of $x$ and $y$, thus obey the Laplace equations:

$$\triangle s = \frac{\partial^2 s}{\partial x^2} + \frac{\partial^2 s}{\partial y^2} = s_{xx} + s_{yy} = 0, \tag{5}$$

$$\triangle t = \frac{\partial^2 t}{\partial x^2} + \frac{\partial^2 t}{\partial y^2} = t_{xx} + t_{yy} = 0. \tag{6}$$

The two Laplace equations $\triangle s = 0$ and $\triangle t = 0$, together with the above specified boundary conditions, define the mapping $\vec{s} : \mathcal{D} \mapsto \mathcal{P}$. Note that this mapping only depends on the shape of domain $\mathcal{D}$. By interchanging the dependent and independent variables, a non-linear elliptic partial differential equation can be derived for $\vec{x} : \mathcal{P} \mapsto \mathcal{D}$. This mapping is called the elliptic transformation. It is well known that this mapping is differentiable and one-to-one [3].

Define the computational space $\mathcal{C}$ as the unit square in a two dimensional space with Cartesian coordinates $\vec{\xi} = (\xi, \eta)^T$. Assume that a mapping $\vec{x} : \partial\mathcal{C} \mapsto \partial\mathcal{D}$ is prescribed which maps the boundary of $\mathcal{C}$ one-to-one on the boundary of $\mathcal{D}$. This mapping defines the boundary grid point distribution. Assume that

- $\xi \equiv 0$ at edge $E_1$ and $\xi \equiv 1$ at edge $E_2$,
- $\eta \equiv 0$ at edge $E_3$ and $\eta \equiv 1$ at edge $E_4$.

We wish to construct a mapping $\vec{x} : \mathcal{C} \mapsto \mathcal{D}$ which obeys the boundary conditions and which is a differentiable one-to-one mapping. Furthermore, we require that the interior grid point distribution describes a smooth transition between the prescribed grid point distribution in the four edges.

A natural mapping $\vec{x} : \mathcal{C} \mapsto \mathcal{D}$ exists which obeys these requirements. This mapping will be the composition of an algebraic transformation and the elliptic transformation based on the Laplace equations. The algebraic transformation is a differentiable one-to-one mapping from computational space $\mathcal{C}$ onto the parameter space $\mathcal{P}$. The composition of these two mappings defines the interior grid point distribution and is a differentiable one-to-one mapping from computational domain $\mathcal{C}$ onto domain $\mathcal{D}$.

The algebraic transformation is defined as follows. Because $\vec{x} : \partial\mathcal{C} \mapsto \partial\mathcal{D}$ is prescribed and $\vec{x} : \partial\mathcal{P} \mapsto \partial\mathcal{D}$ is defined as described above, it follows that $\vec{s} : \partial\mathcal{C} \mapsto \partial\mathcal{P}$ is also defined.

From the preceding requirements it follows that

$$s(0,\eta) = 0 \ , \ s(1,\eta) = 1 \ , \ s(\xi,0) = s_{E_3}(\xi) \ , \ s(\xi,1) = s_{E_4}(\xi), \tag{7}$$

where the functions $s_{E_3}, s_{E_4}$ are monotonically increasing, and

$$t(\xi,0) = 0 \ , \ t(\xi,1) = 1 \ , \ t(0,\eta) = t_{E_1}(\eta) \ , \ t(1,\eta) = t_{E_2}(\eta), \tag{8}$$

where the functions $t_{E_1}, t_{E_2}$ are also monotonically increasing. Thus the four functions $t_{E_1}(\eta)$, $t_{E_2}(\eta)$, $s_{E_3}(\xi)$, $s_{E_4}(\xi)$ are defined by the boundary grid point distribution.

The mapping $\vec{s} : \mathcal{C} \mapsto \mathcal{P}$ is now defined by the following two algebraic equations:

$$s = s_{E_3}(\xi)(1-t) + s_{E_4}(\xi)t, \tag{9}$$

$$t = t_{E_1}(\eta)(1-s) + t_{E_2}(\eta)s. \tag{10}$$

Eq.(9) implies that a coordinate line $\xi$ = constant is mapped to the parameter space $\mathcal{P}$ as a straight line: $s$ is a linear function of $t$, and Eq.(10) implies that a grid line $\eta$ = constant is also mapped to $\mathcal{P}$ as a straight line: $t$ is a linear function of $s$. For given values of $\xi$ and $\eta$, the corresponding $s$ and $t$ values are found as the intersection point of the two straight lines. For this reason, the system defined by Eqs.(9),(10) is called the "algebraic straight line transformation". It can be easily verified that this system defines a differentiable one-to-one mapping because of the positiveness of the Jacobian: $s_\xi t_\eta - s_\eta t_\xi > 0$.

In the remainder of this section, we will derive the set of non-linear elliptic partial differential equations which the composite mapping $\vec{x} = \vec{x}(\vec{s}(\vec{\xi}))$ has to fulfill. First introduce the two covariant base vectors

$$\vec{a}_1 = \frac{\partial\vec{x}}{\partial\xi} = \vec{x}_\xi \ , \ \vec{a}_2 = \frac{\partial\vec{x}}{\partial\eta} = \vec{x}_\eta, \tag{11}$$

and define the covariant metric tensor components as the inner product of the covariant base vectors

$$a_{ij} = (\vec{a}_i, \vec{a}_j) \ , \ i = \{1,2\} \ , \ j = \{1,2\}. \tag{12}$$

Then the contravariant base vectors $\vec{a}^1$ and $\vec{a}^2$ are defined according to the rules

$$(\vec{a}^i, \vec{a}_j) = \delta^i_j \ , \ i = \{1,2\} \ , \ j = \{1,2\}, \tag{13}$$

with $\delta^i_j$ the Kronecker symbol. Define the contravariant metric tensor components

$$a^{ij} = (\vec{a}^i, \vec{a}^j) \ , \ i = \{1,2\} \ , \ j = \{1,2\}, \tag{14}$$

so that

$$\begin{pmatrix} a_{11} & a_{12} \\ a_{12} & a_{22} \end{pmatrix} \begin{pmatrix} a^{11} & a^{12} \\ a^{12} & a^{22} \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}, \tag{15}$$

and

$$\vec{a}^1 = a^{11}\vec{a}_1 + a^{12}\vec{a}_2 \ , \ \vec{a}^2 = a^{12}\vec{a}_1 + a^{22}\vec{a}_2. \tag{16}$$

Introduce the determinant $J^2$ of the covariant metric tensor: $J^2 = a_{11}a_{22} - a_{12}^2$.

Now consider an arbitrary function $\phi = \phi(\xi, \eta)$. Then $\phi$ is also defined in domain $\mathcal{D}$ and the Laplacian of $\phi$ is expressed as

$$\triangle\phi = \phi_{xx} + \phi_{yy} = \frac{1}{J}\left\{\left(Ja^{11}\phi_\xi + Ja^{12}\phi_\eta\right)_\xi + \left(Ja^{12}\phi_\xi + Ja^{22}\phi_\eta\right)_\eta\right\}, \tag{17}$$

which may be found in every textbook on Tensor Analysis and Differential Geometry (for example see [4], page 227). Take as special cases respectively $\phi \equiv \xi$ and $\phi \equiv \eta$. Then Eq.(17) yields

$$\triangle\xi = \frac{1}{J}\left\{\left(Ja^{11}\right)_\xi + \left(Ja^{12}\right)_\eta\right\}\ ,\ \triangle\eta = \frac{1}{J}\left\{\left(Ja^{12}\right)_\xi + \left(Ja^{22}\right)_\eta\right\}. \tag{18}$$

Thus the Laplacian of $\phi$ can also be expressed as

$$\triangle\phi = a^{11}\phi_{\xi\xi} + 2a^{12}\phi_{\xi\eta} + a^{22}\phi_{\eta\eta} + \triangle\xi\ \phi_\xi + \triangle\eta\ \phi_\eta. \tag{19}$$

Substitution of respectively $\phi \equiv s$ and $\phi \equiv t$ in this equation yields

$$\triangle s = a^{11}s_{\xi\xi} + 2a^{12}s_{\xi\eta} + a^{22}s_{\eta\eta} + \triangle\xi\ s_\xi + \triangle\eta\ s_\eta, \tag{20}$$

$$\triangle t = a^{11}t_{\xi\xi} + 2a^{12}t_{\xi\eta} + a^{22}t_{\eta\eta} + \triangle\xi\ t_\xi + \triangle\eta\ t_\eta. \tag{21}$$

Using these equations and the requirement that $s$ and $t$ are harmonic in domain $\mathcal{D}$, thus $\triangle s = 0$ and $\triangle t = 0$, we find the following expressions for the Laplacian of $\xi$ and $\eta$

$$\begin{pmatrix} \triangle\xi \\ \triangle\eta \end{pmatrix} = a^{11}\vec{P}_{11} + 2a^{12}\vec{P}_{12} + a^{22}\vec{P}_{22}, \tag{22}$$

where

$$\vec{P}_{11} = -T^{-1}\begin{pmatrix} s_{\xi\xi} \\ t_{\xi\xi} \end{pmatrix}\ ,\ \vec{P}_{12} = -T^{-1}\begin{pmatrix} s_{\xi\eta} \\ t_{\xi\eta} \end{pmatrix}\ ,\ \vec{P}_{22} = -T^{-1}\begin{pmatrix} s_{\eta\eta} \\ t_{\eta\eta} \end{pmatrix}, \tag{23}$$

and the matrix $T$ is defined as

$$T = \begin{pmatrix} s_\xi & s_\eta \\ t_\xi & t_\eta \end{pmatrix}. \tag{24}$$

The six coefficients of the vectors $\vec{P}_{11} = (P_{11}^1, P_{11}^2)^T$ , $\vec{P}_{12} = (P_{12}^1, P_{12}^2)^T$ and $\vec{P}_{22} = (P_{22}^1, P_{22}^2)^T$ are so called control functions. These six control functions are completely defined and easily computed for a given algebraic transformation $\vec{s} = \vec{s}(\vec{\xi})$. Different and less useful expressions of these control functions can also be found in [5, 6].

Finally, substitution of $\phi \equiv \vec{x}$ in Eq.(19) yields

$$\triangle\vec{x} = a^{11}\vec{x}_{\xi\xi} + 2a^{12}\vec{x}_{\xi\eta} + a^{22}\vec{x}_{\eta\eta} + \triangle\xi\ \vec{x}_\xi + \triangle\eta\ \vec{x}_\eta. \tag{25}$$

Substitution of Eq.(22) into this equation and using the fact that $\triangle\vec{x} \equiv 0$ we arrive at the following Poisson grid generation system

$$\begin{aligned} a^{11}\vec{x}_{\xi\xi} + 2a^{12}\vec{x}_{\xi\eta} + a^{22}\vec{x}_{\eta\eta} &+ \left(a^{11}P_{11}^1 + 2a^{12}P_{12}^1 + a^{22}P_{22}^1\right)\vec{x}_\xi \\ &+ \left(a^{11}P_{11}^2 + 2a^{12}P_{12}^2 + a^{22}P_{22}^2\right)\vec{x}_\eta = 0. \end{aligned} \tag{26}$$

Using Eqs.(12),(15), we find the following well known expressions for the contravariant metric tensor components:

$$J^2 a^{11} = a_{22} = (\vec{x}_\eta, \vec{x}_\eta)\ ,\ J^2 a^{12} = -a_{12} = -(\vec{x}_\xi, \vec{x}_\eta)\ ,\ J^2 a^{22} = a_{11} = (\vec{x}_\xi, \vec{x}_\xi). \tag{27}$$

Thus the Poisson grid generation system defined by Eq.(26) can be simplified by multiplication with $J^2$. Then we obtain:

$$\begin{aligned} a_{22}\vec{x}_{\xi\xi} - 2a_{12}\vec{x}_{\xi\eta} + a_{11}\vec{x}_{\eta\eta} \quad &+ \quad \left(a_{22}P_{11}^1 - 2a_{12}P_{12}^1 + a_{11}P_{22}^1\right)\vec{x}_\xi \\ &+ \quad \left(a_{22}P_{11}^2 - 2a_{12}P_{12}^2 + a_{11}P_{22}^2\right)\vec{x}_\eta = 0. \end{aligned} \tag{28}$$

This equation, together with the expressions for the control functions $P_{ij}^k$ given by Eq.(23), forms our 2D grid generation system. Grids are computed by solving this quasi-linear system of elliptic partial differential equations.

**Orthogonality at boundaries**

Grid orthogonality at boundaries can be achieved as follows.

Redefine the elliptic transformation $\vec{x} : \mathcal{P} \mapsto \mathcal{D}$ by imposing the following new set of boundary conditions for the harmonic functions $s$ and $t$:

- $s \equiv 0$ at edge $E_1$ and $s \equiv 1$ at edge $E_2$,
- $\frac{\partial s}{\partial n} = 0$ along edges $E_3$ and $E_4$, where $n$ is the outward normal direction,
- $t \equiv 0$ at edge $E_3$ and $t \equiv 1$ at edge $E_4$,
- $\frac{\partial t}{\partial n} = 0$ along edges $E_1$ and $E_2$, where $n$ is the outward normal direction.

These new boundary conditions define a new mapping $\vec{x} : \mathcal{P} \mapsto \mathcal{D}$.

The Neumann boundary condition $\frac{\partial s}{\partial n} = 0$ along edges $E_3$ and $E_4$ imply that a parameter line $s =$ constant is a curve in domain $\mathcal{D}$ which is orthogonal at those edges. Similarly, a parameter line $t =$ constant is a curve in $\mathcal{D}$ which is orthogonal at edge $E_1$ and edge $E_2$.

The algebraic transformation $\vec{s} : \mathcal{C} \mapsto \mathcal{P}$ is redefined according to the following two algebraic equations:

$$s = s_{E_3}(\xi)H_0(t) + s_{E_4}(\xi)H_1(t), \tag{29}$$

$$t = t_{E_1}(\eta)H_0(s) + t_{E_2}(\eta)H_1(s), \tag{30}$$

where $H_0$ and $H_1$ are cubic Hermite interpolation functions defined in Eq.(52) below. Note that $H_0(0) = 1$, $H_0'(0) = 0$, $H_0(1) = 0$, $H_0'(1) = 0$ and $H_1(0) = 0$, $H_1'(0) = 0$, $H_1(1) = 1$, $H_1'(1) = 0$. It follows from Eq.(29) that a coordinate line $\xi =$ constant is mapped to parameter space $\mathcal{P}$ as a cubic curve which is orthogonal at both edge $E_3$ and edge $E_4$ in $\mathcal{P}$. Such a curve in parameter space $\mathcal{P}$ will thus be mapped by the new elliptic transformation $\vec{x} : \mathcal{P} \mapsto \mathcal{D}$ as a curve which is orthogonal at both edge $E_3$ and edge $E_4$ in $\mathcal{D}$. Similar observations can be made for coordinate lines $\eta =$ constant. Thus the grid will be orthogonal at all four edges in domain $\mathcal{D}$.

The composite mapping $\vec{x} : \mathcal{C} \mapsto \mathcal{D}$ still obeys the Poisson grid generation system defined by Eq.(28). Thus the same system of elliptic equations can be solved to generate an orthogonal grid at the boundary. The only difference is that now $\vec{s} : \mathcal{C} \mapsto \mathcal{P}$ is defined by Eqs.(29),(30) instead of Eqs.(9),(10).

Figs.5,6,7 are demonstrations of the robustness of this elliptic grid generation method. The boundary grid point distribution is prescribed and the interior grids are obtained by solving Eq.(28). The interior grid point distributions were verified to be foldfree by zooming into regions where the grid is very dense.

## 3 Surface Grid Generation on Minimal Surfaces

Grid generation on a minimal surface in 3D physical space is in fact equivalent to grid generation in a domain in 2D physical space.

As in the two dimensional case, again consider four curved edges $E_1, E_2, E_3, E_4$ but now situated in the three dimensional physical space with Cartesian coordinates $\vec{x} = (x, y, z)^T$. Let $(E_1, E_2)$ and $(E_3, E_4)$ be the two pairs of opposite edges as shown in Fig.2.

Introduce the parameter space $\mathcal{P}$ as the unit square in a two dimensional space with Cartesian coordinates $\vec{s} = (s, t)^T$. Again require that the parameters $s$ and $t$ obey the boundary equations specified in Eqs.(1),(2),(3),(4). Furthermore, require that

$$\triangle s = 0, \tag{31}$$

$$\triangle t = 0, \tag{32}$$

$$H = 0, \tag{33}$$

where $\triangle$ is the Laplace-Beltrami operator for surfaces and $H$ is the mean curvature.

These three requirements, together with the specified boundary conditions define a unique mapping $\vec{x} : \mathcal{P} \mapsto \mathcal{R}^3$. The shape of the surface defined by this mapping is a minimal surface (soap film) because of the requirement that the mean curvature $H$ is zero. The parametrization of the surface is defined by Eqs.(31),(32). Define the minimal surface $\mathcal{S} = \{\vec{x}(s, t) \mid (s, t) \in \mathcal{P}\}$.

Consider a prescribed boundary grid point distribution at the four edges $E_1, E_2, E_3, E_4$ of the minimal surface $\mathcal{S}$. The boundary grid point distribution can be defined as a mapping $\vec{x} : \partial\mathcal{C} \mapsto \partial\mathcal{S}$ where $\mathcal{C}$ is the computational space defined as the unit square in a two dimensional space with Cartesian coordinates $\vec{\xi} = (\xi, \eta)^T$. Because $\vec{x} : \partial\mathcal{C} \mapsto \partial\mathcal{S}$ is prescribed and $\vec{x} : \partial\mathcal{P} \mapsto \partial\mathcal{S}$ is defined as described above, it follows that $\vec{s} : \partial\mathcal{C} \mapsto \partial\mathcal{P}$ is also defined.

In exactly the same way as for the two dimensional case, the mapping $\vec{s} : \mathcal{C} \mapsto \mathcal{P}$ is defined by the algebraic straight line transformation defined by Eqs.(9),(10). The mapping $\vec{x} : \mathcal{P} \mapsto \mathcal{S}$ is defined by Eqs.(31),(32),(33). The composite mapping $\vec{x} : \mathcal{C} \mapsto \mathcal{S}$ is defined as $\vec{x} = \vec{x}(\vec{s}(\vec{\xi}))$ and describes the interior grid point distribution on the minimal surface $\mathcal{S}$. Note that this composite mapping will be differentiable and one-to-one.

We will now show that the set of non-linear elliptic partial differential equations which the composite mapping has to fulfill is the same Poisson system as defined by Eq.(28) but with $\vec{x} = (x, y, z)^T$ instead of $\vec{x} = (x, y)^T$. Thus grid generation on a minimal surface in 3D physical space is in fact

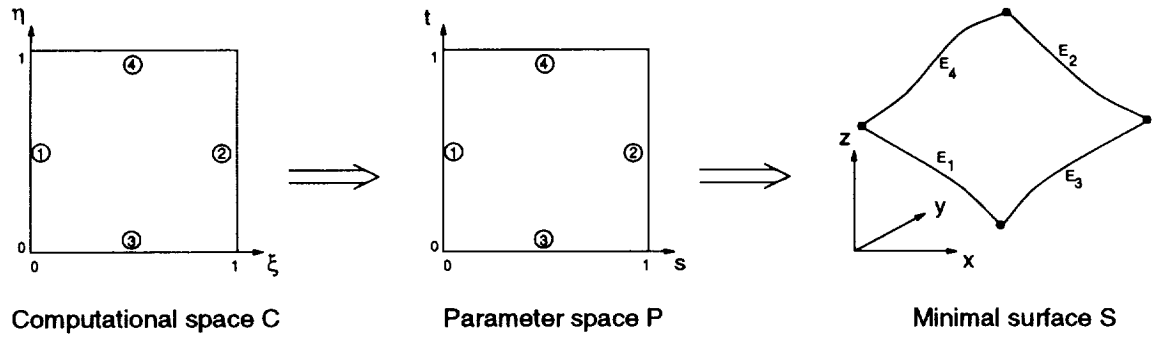



Figure 2: Transformation from computational $(\xi, \eta)$ space to a minimal surface $\mathcal{S}$ in Cartesian $(x, y, z)$ space.

equivalent to grid generation in a domain in 2D physical space. The result that a Poisson system of the form as defined by Eq.(28) can be used to compute a grid on a minimal surface can also be found as a special application of the formulas derived in [7].

For this purpose, introduce the two covariant base vectors

$$\vec{a}_1 = \vec{x}_\xi \ , \ \vec{a}_2 = \vec{x}_\eta. \tag{34}$$

The two covariant base vectors span the tangent plane of $\mathcal{S}$ at the corresponding point $P$. Define the unit surface normal as

$$\vec{n} = \frac{\vec{a}_1 \wedge \vec{a}_2}{\| \vec{a}_1 \wedge \vec{a}_2 \|}, \tag{35}$$

where $\wedge$ is the vector product operator. The contravariant base vectors $\vec{a}^1$ and $\vec{a}^2$ are defined according to the rules

$$(\vec{a}^i, \vec{a}_j) = \delta^i_j \ , \ i = \{1,2\} \ , \ j = \{1,2\}, \tag{36}$$

and

$$(\vec{a}^1, \vec{n}) = 0 \ , \ (\vec{a}^2, \vec{n}) = 0. \tag{37}$$

Thus the two contravariant base vectors are also lying in the tangent plane of $\mathcal{S}$ at the corresponding point $P$. Define the covariant metric tensor components by Eq.(12) and the contravariant metric tensor components by Eq.(14). Then Eqs.(15),(16) are still valid. Again introduce the determinant $J^2$ of the covariant metric tensor: $J^2 = a_{11}a_{22} - a_{12}^2$.

Now consider an arbitrary function $\phi = \phi(\xi, \eta)$. Then $\phi$ is also defined on surface $\mathcal{S}$ and the Laplace-Beltrami operator of $\phi$ is expressed as

$$\triangle\phi = \frac{1}{J}\left\{\left(Ja^{11}\phi_\xi + Ja^{12}\phi_\eta\right)_\xi + \left(Ja^{12}\phi_\xi + Ja^{22}\phi_\eta\right)_\eta\right\} \tag{38}$$

(see [4], page 227). As in the two-dimensional case, substitution of $\phi \equiv \xi$ and $\phi \equiv \eta$ into this equation yields Eq.(18). Thus the Laplace-Beltrami operator of $\phi$ can also be expressed as defined by Eq.(19). Substitution of respectively $\phi \equiv s$ and $\phi \equiv t$ in Eq.(19) and using the requirements expressed by Eqs.(31),(32) yields exactly the same expressions for $\triangle\xi$ and $\triangle\eta$ given by Eqs.(22),(23). Finally, substitution of $\phi \equiv \vec{x}$ in Eq.(19) yields Eq.(25).

The Laplace-Beltrami operator applied on $\vec{x}$ obeys a famous relation expressed by

$$\triangle \vec{x} = 2H\vec{n}, \tag{39}$$

where the mean curvature $H$ is defined as

$$H = \frac{1}{2}\left(a^{11}\vec{x}_{\xi\xi} + 2a^{12}\vec{x}_{\xi\eta} + a^{22}\vec{x}_{\eta\eta}, \vec{n}\right). \tag{40}$$

(for example see [8], Theorem 1, page 71). Using the requirement $H = 0$ yields

$$\triangle \vec{x} = 0. \tag{41}$$

Thus Eq.(22) and Eq.(25) with $\triangle \vec{x} = 0$ are also valid for minimal surfaces. Following the same derivation as given in Section 2, we arrive at exactly the same non-linear system of elliptic partial differential equations as expressed by Eq.(28). Thus an interior grid point distribution on a minimal surface is found by solving Eq.(28). The only difference compared to the two dimensional case is that now $\vec{x} = (x, y, z)^T$ instead of $\vec{x} = (x, y)^T$ .

Grid orthogonality at boundaries can be obtained in the same way as described in Section 2.

One may ask whether it is useful to implement a method to compute grids on minimal surfaces in a 3D multi-block grid generator code. The answer is yes. Minimal surfaces may be used to define the geometry and grid for a block-face of which only the four face-edges are given. It is also possible to apply minimal surface grid generation when a grid must be generated in a block-face with four face-edges lying in a plane. Then the minimal surface is a plane surface bounded by the four edges. The grids in the 2D domains depicted in Figs.5,6,7 were generated in this way and are in fact grids on minimal surfaces.

An example of a grid on a characteristic minimal surface is shown in Fig.8. This is a so-called square Scherck surface [8]. Fig.9 illustrates what happens when the prescribed boundary grid point distribution is changed. This figure clearly shows that the shape of the minimal surface is independent of the prescribed boundary grid point distribution.

## 4 Surface Grid Generation on Parametrized Surfaces

In this section we develop a method to generate a grid on a parametrized surface which is independent of the parametrization. A generated grid only depends on the shape of the surface and the prescribed boundary grid point distribution at the four edges of the surface.

Consider a bounded surface $\mathcal{S}$ with a prescribed geometrical shape in three dimensional physical space with Cartesian coordinates $\vec{x} = (x, y, z)^T$. Assume that $\mathcal{S}$ is parametrized by a differentiable one-to-one mapping

$$\vec{x} : \mathcal{P}_{uv} \mapsto \mathcal{S}, \tag{42}$$

where $\mathcal{P}_{uv}$ is the unit square in two dimensional space with Cartesian coordinates $\vec{u} = (u, v)^T$. Define the four edges $E_1, E_2, E_3, E_4$ of surface $\mathcal{S}$ by

- $u \equiv 0$ at edge $E_1$ and $u \equiv 1$ at edge $E_2$,

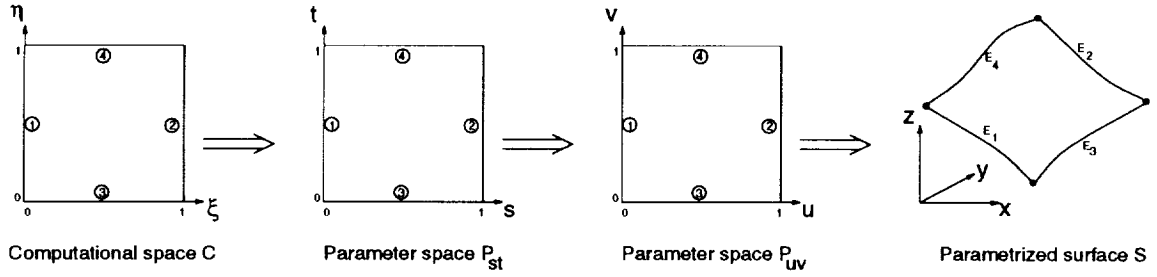



Figure 3: Transformation from computational $(\xi, \eta)$ space to a parametrized surface $\mathcal{S}$ in Cartesian $(x, y, z)$ space.

- $v \equiv 0$ at edge $E_3$ and $v \equiv 1$ at edge $E_4$.

Thus $(E_1, E_2)$ and $(E_3, E_4)$ are the two pairs of opposite edges of surface $\mathcal{S}$ as shown in Fig.3. Introduce the parameter space $\mathcal{P}_{st}$ as the unit square in a two dimensional space with Cartesian coordinates $\vec{s} = (s, t)^T$. Again require that the parameters $s$ and $t$ obey the boundary equations specified in Eqs.(1),(2),(3),(4). Furthermore, require that $\triangle s = 0$ and $\triangle t = 0$ where $\triangle$ is the Laplace-Beltrami operator for surfaces. Hence the parameters $s$ and $t$ obey

$$\left(Ja^{11}s_u + Ja^{12}s_v\right)_u + \left(Ja^{12}s_u + Ja^{22}s_v\right)_v = 0, \tag{43}$$

$$\left(Ja^{11}t_u + Ja^{12}t_v\right)_u + \left(Ja^{12}t_u + Ja^{22}t_v\right)_v = 0, \tag{44}$$

where $a^{ij}$ are the contravariant tensor components and $J^2$ is defined as the determinant of the covariant metric tensor. The contravariant tensor components $a^{ij}$ are related to the covariant tensor components $a_{ij}$ according to Eq.(15). The covariant metric tensor components are defined by Eq.(12), where the two covariant base vectors are now given by

$$\vec{a}_1 = \vec{x}_u \ , \ \vec{a}_2 = \vec{x}_v \ . \tag{45}$$

The coefficients $Ja^{11}$, $Ja^{12}$ and $Ja^{22}$ in Eqs.(43),(44) are thus functions of $u$ and $v$, and Eqs.(43),(44) are therefore two uncoupled second-order linear partial differential equations for the functions $s = s(u, v)$ and $t = t(u, v)$.

Each boundary point of surface $\mathcal{S}$ has a unique $(s, t)$ parameter value at $\partial\mathcal{P}_{st}$ and a unique $(u, v)$ parameter value at $\partial\mathcal{P}_{uv}$. Thus each $(u, v)$ parameter value at $\partial\mathcal{P}_{uv}$ has also a unique $(s, t)$ parameter value at $\partial\mathcal{P}_{st}$. Thus the functions $s$ and $t$ are prescribed at the boundary of $\mathcal{P}_{uv}$. Hence, Eq.(43) together with the Dirichlet boundary conditions for $s$ can be used to compute $s = s(u, v)$ and Eq.(44) together with the Dirichlet boundary conditions for $t$ can be used to compute $t = t(u, v)$. Only two linear partial differential equations have to be solved to define these mappings. These two mappings are compactly written as $\vec{s} : \mathcal{P}_{uv} \mapsto \mathcal{P}_{st}$. Note that $\vec{s} : \mathcal{P}_{uv} \mapsto \mathcal{P}_{st}$ is a differentiable one-to-one mapping so that the inverse mapping $\vec{u} : \mathcal{P}_{st} \mapsto \mathcal{P}_{uv}$ also exists.

Thus the composite mapping $\vec{x} : \mathcal{P}_{st} \mapsto \mathcal{S}$, defined as $\vec{x} = \vec{x}(\vec{u}(\vec{s}))$ also exists and is differentiable and one-to-one. Note that this mapping $\vec{x} : \mathcal{P}_{st} \mapsto \mathcal{S}$ only depends on the shape of surface $\mathcal{S}$ and is independent of the original parametrization $\vec{x} : \mathcal{P}_{uv} \mapsto \mathcal{S}$. The mapping $\vec{x} : \mathcal{P}_{st} \mapsto \mathcal{S}$ may thus be considered as a property of surface $\mathcal{S}$ and defines a new unique parametrization of $\mathcal{S}$.

Consider a prescribed boundary grid point distribution at the four edges $E_1, E_2, E_3, E_4$. The boundary grid point distribution can be defined as a mapping $\vec{x} : \partial\mathcal{C} \mapsto \partial\mathcal{S}$ where $\mathcal{C}$ is the computational space defined as the unit square in a two dimensional space with Cartesian coordinates $\vec{\xi} = (\xi, \eta)^T$. Because $\vec{x} : \partial\mathcal{C} \mapsto \partial\mathcal{S}$ is prescribed and $\vec{x} : \partial\mathcal{P}_{st} \mapsto \partial\mathcal{S}$ is defined as described above, it follows that $\vec{s} : \partial\mathcal{C} \mapsto \partial\mathcal{P}_{st}$ is also defined.

In exactly the same way as for the two dimensional case, the mapping $\vec{s} : \mathcal{C} \mapsto \mathcal{P}_{st}$ is now defined by the algebraic straight line transformation defined by Eqs.(9),(10). The composition of the mapping $\vec{s} : \mathcal{C} \mapsto \mathcal{P}_{st}$ and the mapping $\vec{x} : \mathcal{P}_{st} \mapsto \mathcal{S}$ defines $\vec{x} : \mathcal{C} \mapsto \mathcal{S}$ and describes the interior grid point distribution on surface $\mathcal{S}$. Note that this composite mapping will also be differentiable and one-to-one.

Grid orthogonality at boundaries can be obtained by the same procedure as described in Section 2 for 2D domains.

Fig.10 shows an irregularly distributed control point mesh on a smooth surface. The surface is defined as $z = \frac{1}{8}tanh(15(\frac{1}{4} - (x-1)^2 - (y-1)^2)), (x, y) \in [0, 1]^2$. In section 5.2 is described how bicubic Hermite interpolation is used to define the mapping $\vec{x} : \mathcal{P}_{uv} \mapsto \mathcal{S}$. The parametrization depends on the control point distribution. Fig.11 shows an elliptic grid. Equidistributed boundary grid points are used as Dirichlet boundary condition. This figure clearly demonstrates that the interior surface grid only depends on the shape of the surface and is independent of the parametrization.

Less academic surface grids are shown in Fig.13.

## 5 Surface modeling

Consider a discrete surface defined as a two-dimensional array of control points. An interpolated surface is obtained by bicubic Hermite interpolation. Non-linear averaging formulas for the tangent and twist vectors are used to prevent spurious oscillations. The interpolation method is explained first for curves and then extended to surfaces.

### 5.1 Piecewise cubic Hermite interpolation for curves

Consider a set of control points $\{\vec{x}_i = (x, y, z)_i^T \mid i = 0 \ldots N\}$. We wish to construct a smooth $C^1$ curve $\vec{x} : u \in [0, 1] \mapsto \mathcal{R}^3$ which is passing through the set of control points with a geometrical shape as one would intuitively expect. Furthermore, spurious oscillations should be prevented. For this reason, cubic spline interpolation is not safe. Instead, piecewise cubic Hermite interpolation is applied. The extra freedom to model the tangent vectors is used to prevent unwanted oscillations. The parameter $u$ is defined as normalized arc length.

Compute the distance between succeeding control points:

$$\bar{d}_i = \| \vec{x}_i - \vec{x}_{i-1} \| , \; i = 1 \ldots N. \tag{46}$$

Define the length of the curve by

$$L = \sum_{i=1}^{N} \bar{d}_i, \tag{47}$$

and the normalized distances as

$$d_i = \bar{d}_i / L \ , \ i = 1 \ldots N. \tag{48}$$

Define the knot sequence $\{u_i \mid i = 0 \ldots N\}$ by $u_0 = 0$ and

$$u_i = u_{i-1} + d_i \ , \ i = 1 \ldots N. \tag{49}$$

Hence, $0 = u_0 < u_1 < \ldots < u_N = 1$. Patch $i$ is defined as the interval $[u_{i-1}, u_i]$. In patch $i$, we relate a local variable $\alpha \in [0, 1]$ to the global variable $u$ by

$$u = u_{i-1} + \alpha(u_i - u_{i-1}) = u_{i-1} + \alpha d_i. \tag{50}$$

For the moment, suppose that the tangent vectors $\{\vec{x}_{u_i} = \frac{d\vec{x}}{du}(u_i), i = 0 \ldots N\}$ are known. How these tangent vectors are modeled is shown below.

The shape of the curve at patch $i$ is then defined as

$$\vec{x}(\alpha) = \vec{x}_{i-1} H_0(\alpha) + \vec{x}_i H_1(\alpha) + d_i \vec{x}_{u_{i-1}} H_2(\alpha) + d_i \vec{x}_{u_i} H_3(\alpha), \tag{51}$$

where $H_0, H_1, H_2, H_3$ are cubic Hermite interpolation functions defined as

$$\begin{aligned} H_0(\alpha) &= (1 + 2\alpha)(1 - \alpha)^2, \\ H_1(\alpha) &= (3 - 2\alpha)\alpha^2, \\ H_2(\alpha) &= \alpha(1 - \alpha)^2, \\ H_3(\alpha) &= (\alpha - 1)\alpha^2, \end{aligned} \tag{52}$$

with $0 \leq \alpha \leq 1$.

It can be easily verified that $\frac{d\vec{x}}{du}(u_i-) = \frac{d\vec{x}}{du}(u_i+) = \vec{x}_{u_i}$, so that the piecewise cubic curve $\vec{x}(u)$ is indeed $C^1$.

The tangent vectors $\{\vec{x}_{u_i}, i = 0 \ldots N\}$ are computed as follows. Define the chord vectors

$$\vec{x}_{u_{i-\frac{1}{2}}} = \frac{\vec{x}_i - \vec{x}_{i-1}}{d_i} \ , \ i = 1 \ldots N. \tag{53}$$

Note that $\| \vec{x}_{u_{i-\frac{1}{2}}} \| = L$. The tangent vectors at the interior knots $i = 1 \ldots N - 1$ are modeled as

$$\vec{x}_{u_i} = \vec{x}_{u_{i-\frac{1}{2}}} c_i + \vec{x}_{u_{i+\frac{1}{2}}} (1 - c_i) \ , \ i = 1 \ldots N - 1, \tag{54}$$

with

$$c_i = \frac{\| \vec{x}_i - \vec{x}_{i-1} \|^2}{\| \vec{x}_i - \vec{x}_{i-1} \|^2 + \| \vec{x}_{i+1} - \vec{x}_i \|^2} = \frac{d_i^2}{d_i^2 + d_{i+1}^2} \ , \ i = 1 \ldots N - 1. \tag{55}$$

If $\| \vec{x}_i - \vec{x}_{i-1} \| \ll \| \vec{x}_{i+1} - \vec{x}_i \|$ then $c_i \approx 0$ and $\vec{x}_{u_i} \approx \vec{x}_{u_{i+\frac{1}{2}}}$. This implies that high curvature will be restricted to small patches which is a behaviour as one would intuitively expect. Spurious oscillations are also prevented.

Quadratic end conditions are used to compute the end tangent vectors $\vec{x}_{u_0}$ and $\vec{x}_{u_N}$. The quadratic end conditions require that the cubic polynomial function $\vec{x}(\alpha)$ is a quadratic function of $\alpha$ in patch 1 and in patch $N$. It is easily verified that this implies that

$$\vec{x}_{u_0} = 2\vec{x}_{u_{\frac{1}{2}}} - \vec{x}_{u_1} \ , \ \vec{x}_{u_N} = 2\vec{x}_{u_{N-\frac{1}{2}}} - \vec{x}_{u_{N-1}}. \tag{56}$$

Fig.12 illustrates that cubic Hermite interpolation prevents spurious oscillations, in contrast to cubic spline interpolation.

## 5.2 Piecewise bicubic Hermite interpolation for surfaces

Consider a set of control points $\{\vec{x}_{i,j} = (x,y,z)^T_{i,j} \mid i = 0 \ldots N, j = 0 \ldots M\}$. We wish to construct a smooth $C^1$ surface $\vec{x} : (u,v) \in [0,1]^2 \mapsto \mathcal{R}^3$ which is passing through the set of control points with a geometrical shape as one would intuitively expect. As for curves, spurious oscillations should be prevented. For this reason, bicubic spline interpolation is not safe. Instead, piecewise bicubic Hermite interpolation is applied. The extra freedom to model the tangent vectors and twist vectors is used to prevent unwanted oscillations.

Consider a row of points $\{\vec{x}_{i,j} \mid i = 0 \ldots N\}$ with $j \in \{0 \ldots M\}$ fixed. This row of points is considered as a discrete curve and it is therefore possible to compute a knot sequence $\{u_{i,j} \mid i = 0 \ldots N\}$ in exactly the same way as described in the previous section. In the same way, consider a column of points $\{\vec{x}_{i,j} \mid j = 0 \ldots M\}$ with $i \in \{0 \ldots N\}$ fixed, and compute the knot sequence $\{v_{i,j} \mid j = 0 \ldots M\}$.

To construct a smooth surface, one knot sequence is needed for all rows and all columns. These two knot sequences are obtained by averaging:

$$u_i = \frac{1}{M+1}\sum_{j=0}^{M} u_{i,j} \ , \ i = 0 \ldots N, \ v_j = \frac{1}{N+1}\sum_{i=0}^{N} v_{i,j} \ , \ j = 0 \ldots M. \tag{57}$$

Patch $(i,j)$ is defined as the rectangle $[u_{i-1}, u_i] \times [v_{j-1}, v_j]$. In patch $(i,j)$ we relate local variables $(\alpha, \beta) \in [0,1]^2$ to the global variables $(u,v)$ by

$$u = u_{i-1} + \alpha d^u_i \ , \ v = v_{i-1} + \beta d^v_j, \tag{58}$$

with $d^u_i = u_i - u_{i-1}$ and $d^v_j = v_j - v_{j-1}$.

For the moment, suppose that the tangent vectors $\vec{x}_{u_{i,j}} = \frac{\partial \vec{x}}{\partial u}(u_i, v_j)$ , $\vec{x}_{v_{i,j}} = \frac{\partial \vec{x}}{\partial v}(u_i, v_j)$ , and twist vectors $\vec{x}_{uv_{i,j}} = \frac{\partial^2 \vec{x}}{\partial u \partial v}(u_i, v_j)$ are known for all knots $(i,j)$. How these tangent and twist vectors are modeled is shown below.

The shape of the surface at patch $(i,j)$ is then defined as

$$\vec{x}(\alpha,\beta) = (H_0(\alpha), H_1(\alpha), H_2(\alpha), H_3(\alpha)) M^H_{i,j} \begin{pmatrix} H_0(\beta) \\ H_1(\beta) \\ H_2(\beta) \\ H_3(\beta) \end{pmatrix}, \tag{59}$$

where the matrix $M^H_{i,j}$ is defined by

$$M^H_{i,j} = \begin{pmatrix} \vec{x}_{i-1,j-1} & \vec{x}_{i-1,j} & d^v_j \vec{x}_{v_{i-1,j-1}} & d^v_j \vec{x}_{v_{i-1,j}} \\ \vec{x}_{i,j-1} & \vec{x}_{i,j} & d^v_j \vec{x}_{v_{i,j-1}} & d^v_j \vec{x}_{v_{i,j}} \\ d^u_i \vec{x}_{u_{i-1,j-1}} & d^u_i \vec{x}_{u_{i-1,j}} & d^u_i d^v_j \vec{x}_{uv_{i-1,j-1}} & d^u_i d^v_j \vec{x}_{uv_{i-1,j}} \\ d^u_i \vec{x}_{u_{i,j-1}} & d^u_i \vec{x}_{u_{i,j}} & d^u_i d^v_j \vec{x}_{uv_{i,j-1}} & d^u_i d^v_j \vec{x}_{uv_{i,j}} \end{pmatrix}. \tag{60}$$

From these definitions, it can be easily verified that the piecewise bicubic surface $\vec{x}(u,v)$ is $C^1$.

The tangent vectors $\vec{x}_{u_{i,j}}$ are computed as follows ( the tangent vectors $\vec{x}_{v_{i,j}}$ are computed in the same way ). Define the chord vectors

$$\vec{x}_{u_{i-\frac{1}{2},j}} = \frac{\vec{x}_{i,j} - \vec{x}_{i-1,j}}{d_i^u} \ , \ i = 1 \ldots N \ , \ j = 0 \ldots M, \tag{61}$$

and use the same non-linear averaging formula as used for curves, thus

$$\vec{x}_{u_{i,j}} = \vec{x}_{u_{i-\frac{1}{2},j}} c_{i,j} + \vec{x}_{u_{i+\frac{1}{2},j}} (1 - c_{i,j}) \ , \ i = 1 \ldots N-1 \ , \ j = 0 \ldots M. \tag{62}$$

with

$$c_{i,j} = \frac{\| \vec{x}_{i,j} - \vec{x}_{i-1,j} \|^2}{\| \vec{x}_{i,j} - \vec{x}_{i-1,j} \|^2 + \| \vec{x}_{i+1,j} - \vec{x}_{i,j} \|^2} \ , \ i = 1 \ldots N-1 \ , \ j = 0 \ldots M. \tag{63}$$

Quadratic end conditions are used again to compute the end tangent vectors.

A modification of Adini's method [9] is used to compute the twist vectors. Consider patch $(i,j)$ with local variables $(\alpha, \beta)$. Assume that the tangent vectors $\vec{x}_u$, $\vec{x}_v$ are known at the four corner points of the patch. Introduce the abbreviate notation $\vec{x}_{00} = \vec{x}_{i-1,j-1}$, $\vec{x}_{10} = \vec{x}_{i,j-1}$, $\vec{x}_{01} = \vec{x}_{i-1,j}$, $\vec{x}_{11} = \vec{x}_{i,j}$. Use Eqs.(59),(60) to find $\vec{x}_\alpha(0,0) = d_i^u \vec{x}_{u_{i-1,j-1}}$, $\vec{x}_\alpha(1,0) = d_i^u \vec{x}_{u_{i,j-1}}$, $\vec{x}_\alpha(0,1) = d_i^u \vec{x}_{u_{i-1,j}}$, $\vec{x}_\alpha(1,1) = d_i^u \vec{x}_{u_{i,j}}$, $\vec{x}_\beta(0,0) = d_j^v \vec{x}_{v_{i-1,j-1}}$, $\vec{x}_\beta(1,0) = d_j^v \vec{x}_{v_{i,j-1}}$, $\vec{x}_\beta(0,1) = d_j^v \vec{x}_{v_{i-1,j}}$, $\vec{x}_\beta(1,1) = d_j^v \vec{x}_{v_{i,j}}$. Compute the boundary curves of the patch by cubic Hermite interpolation. Thus, for example

$$\vec{x}(\alpha, 0) = \vec{x}_{00} H_0(\alpha) + \vec{x}_{10} H_1(\alpha) + \vec{x}_\alpha(0,0) H_2(\alpha) + \vec{x}_\alpha(1,0) H_3(\alpha), \tag{64}$$

and the other three boundary curves are computed by similar formulas. Define the shape of the surface patch as a bilinearly blended Coon's patch

$$\vec{x}(\alpha,\beta) = (1-\alpha, \alpha) \begin{pmatrix} \vec{x}(0,\beta) \\ \vec{x}(1,\beta) \end{pmatrix} + (1-\beta, \beta) \begin{pmatrix} \vec{x}(\alpha,0) \\ \vec{x}(\alpha,1) \end{pmatrix} - (1-\alpha, \alpha) \begin{pmatrix} \vec{x}_{00} & \vec{x}_{01} \\ \vec{x}_{10} & \vec{x}_{11} \end{pmatrix} \begin{pmatrix} 1-\beta \\ \beta \end{pmatrix}. \tag{65}$$

Compute the corner twist vectors $\vec{x}_{\alpha\beta}(0,0), \vec{x}_{\alpha\beta}(1,0), \vec{x}_{\alpha\beta}(0,1), \vec{x}_{\alpha\beta}(1,1)$ from Eq.(65). Use Eq.(58) to find the corresponding twist vectors w.r.t. the global variables $(u,v)$: $\vec{x}_{uv_{i-1,j-1}} = \vec{x}_{\alpha\beta}(0,0)/d_i^u d_j^v$ etc..

Thus at the four corners of each patch, an estimation is found for the twist vector $\vec{x}_{uv}$. Consider an interior knot $(i,j)$. Then four estimations for the twist vector $\vec{x}_{uv_{i,j}}$ are found in respectively patches $(i,j)$,$(i+1,j)$,$(i,j+1)$ and $(i+1,j+1)$. Write those estimations as respectively $\vec{x}_{uv}^{SW}$, $\vec{x}_{uv}^{SE}$, $\vec{x}_{uv}^{NW}$, $\vec{x}_{uv}^{NE}$. A similar averaging procedure as applied for tangent vectors is used to define a unique value $\vec{x}_{uv_{i,j}}$:

$$\vec{x}_{uv_{i,j}} = \frac{\vec{x}_{uv}^{SW} A_{i,j}^2 + \vec{x}_{uv}^{SE} A_{i+1,j}^2 + \vec{x}_{uv}^{NW} A_{i,j+1}^2 + \vec{x}_{uv}^{NE} A_{i+1,j+1}^2}{A_{i,j}^2 + A_{i+1,j}^2 + A_{i,j+1}^2 + A_{i+1,j+1}^2}, \tag{66}$$

where $A_{i,j}$ is the area of patch $(i,j)$ defined as

$$A_{i,j} = 0.5 \, \| \, (\vec{x}_{i,j} - \vec{x}_{i-1,j-1}) \wedge (\vec{x}_{i-1,j} - \vec{x}_{i,j-1}) \, \| \, . \tag{67}$$

This non-linear averaging procedure guarantees that large changes in twist will be restricted to small patches. At a boundary knot, there are only two estimations for the twist vector. It is evident how the averaging procedure must be applied in that case. At the four corner knots, only one estimation is available and averaging is thus not needed.

Figs.10,11 illustrate that bicubic Hermite interpolation gives a smooth surface shape, even if the control points are irregularly distributed.

## Concluding remarks

An elliptic grid generation method is presented to generate boundary conforming grids in domains in 2D physical space and on minimal surfaces and parametrized surfaces in 3D physical space. The method is based on the use of composite mappings and produce excellent grids in the sense of smoothness, grid point distribution and regularity.

The described elliptic grid generation method has been implemented into NLR's multi-block grid generation code ENGRID and is extensively used for the generation of boundary conforming Navier-Stokes grids in block-faces with complex shapes.

Concerning surface modeling, it is shown that bicubic Hermite interpolation is an excellent method to define interpolated surfaces.

## References


[1] J.W. Boerstoel, S.P. Spekreijse, P.L. Vitagliano, The Design of a System of Codes for Industrial Calculations of Flows around Aircraft and other Complex Aerodynamic Configurations, NLR-TP-92190, AIAA-92-2619-CP, 10th Applied Aerodynamics Conference, Palo Alto, 1992.

[2] S.P. Spekreijse, Elliptic Grid Generation Based on Laplace Equations and Algebraic Transformations, NLR-TP-94102,1994. (Accepted for publication in Journal of Computational Physics).

[3] C. W. Mastin and J. F. Thompson, Transformation of Three-Dimensional Regions onto Rectangular Regions by Elliptic Systems, Numerische Mathematik, vol.29,pp.397-407,1978.

[4] E. Kreyszig, Introduction to Differential Geometry and Riemannian Geometry, Mathematical Expositions no.16, University of Toronto Press,1967.

[5] J. F. Thompson, Z. U. A. Warsi, and C. W. Mastin, Numerical Grid Generation: Foundations and Applications, Elsevier, New York, 1985.

[6] Z.U.A. Warsi, Basic Differential Models for Coordinate Generation, in: Numerical Grid Generation, Ed. J.F. Thompson, Elsevier, North-Holland,pp.41-70,1982.

[7] Z.U.A. Warsi, Numerical Grid Generation in Arbitrary Surfaces through a Second-Order Differential-Geometric Model, Journal of Computational Physics 64,pp.82-96,1986.

[8] U.Dierkes, S. Hildebrandt, A. Kuster, O. Wohlrab, Minimal Surfaces I, Grundlehren der Mathematischen Wissenschaften 295, Springer Verlag, Berlin, 1991.

[9] F.Yamaguchi, Curves and Surfaces in Computer Aided Geometric Design, Springer Verlag, Berlin, 1988.

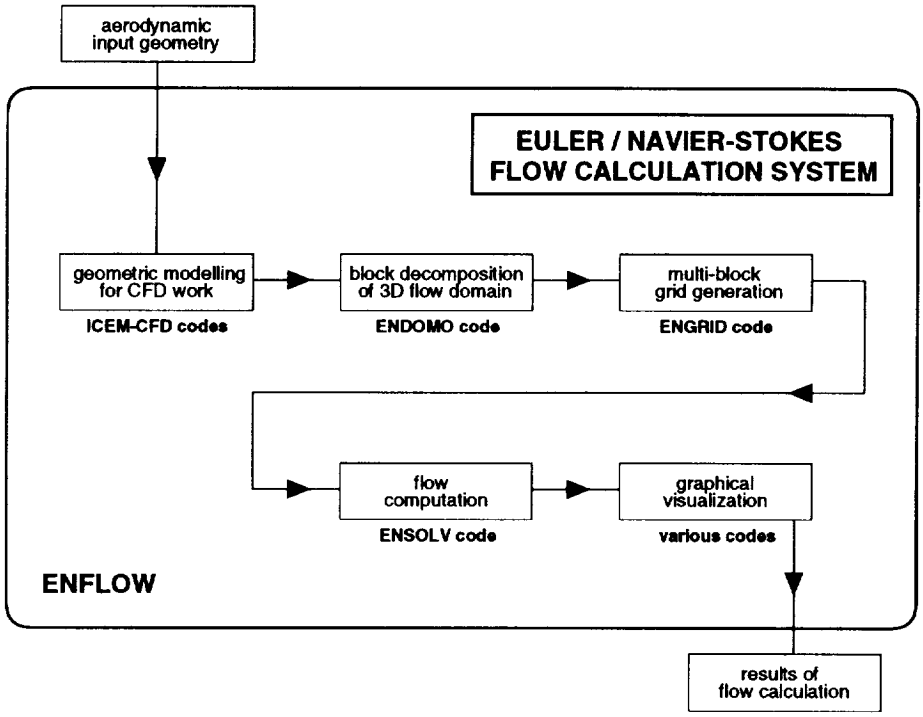



Figure 4: The NLR ENFLOW system

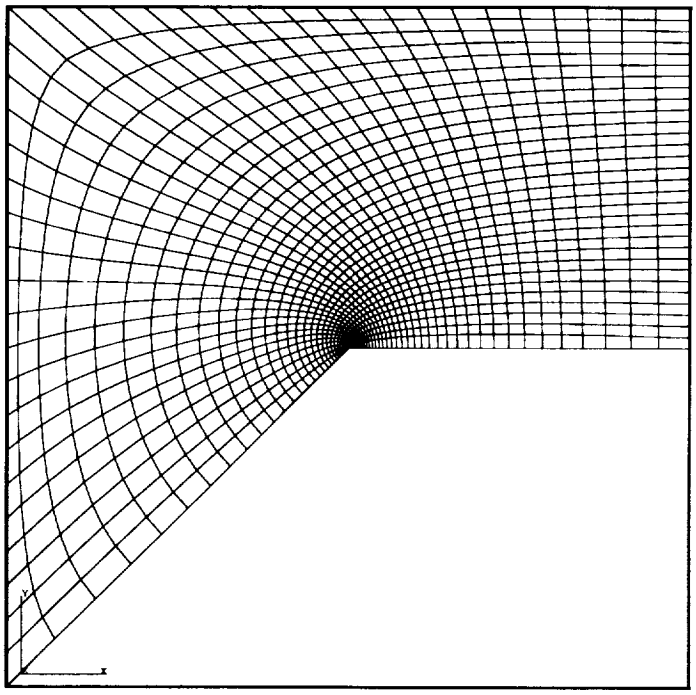


Figure 5: Elliptic grid with grid orthogonality at the lower boundary of the domain.

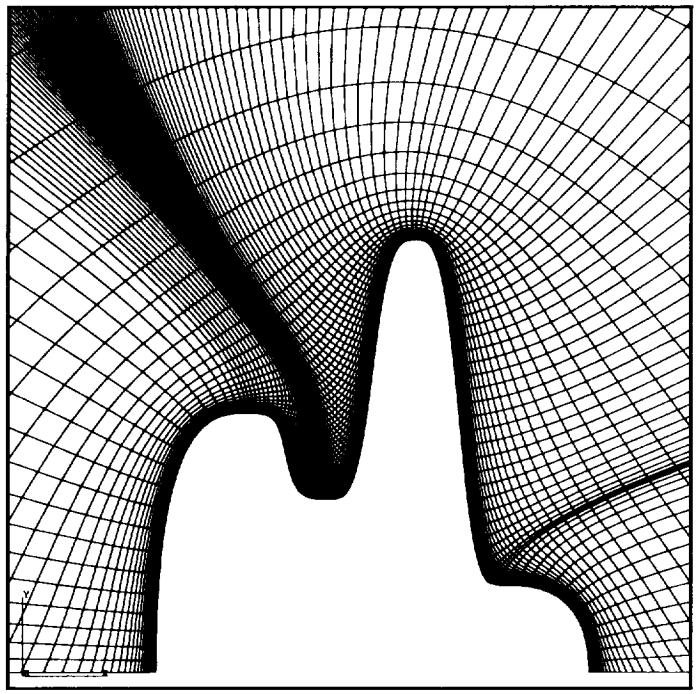


Figure 6: Elliptic grid with boundary layer and orthogonality.

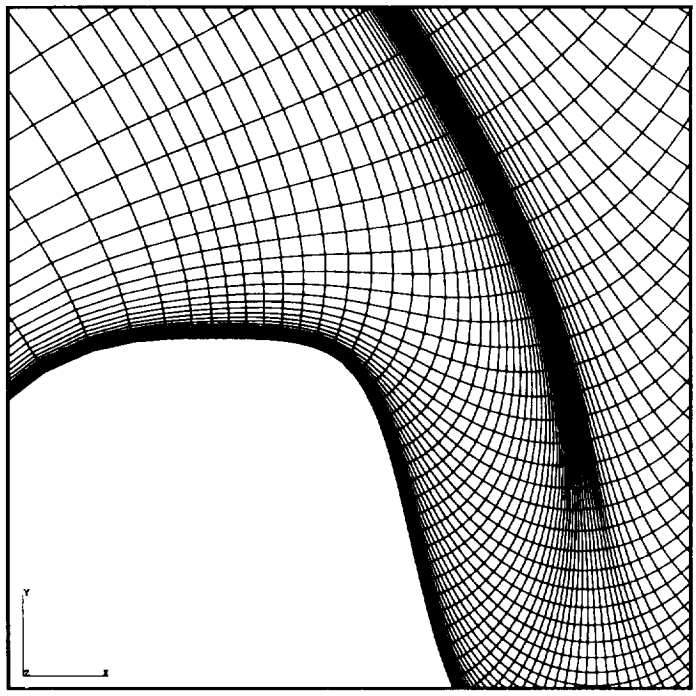


Figure 7: Detail of elliptic grid at convex part of the boundary.

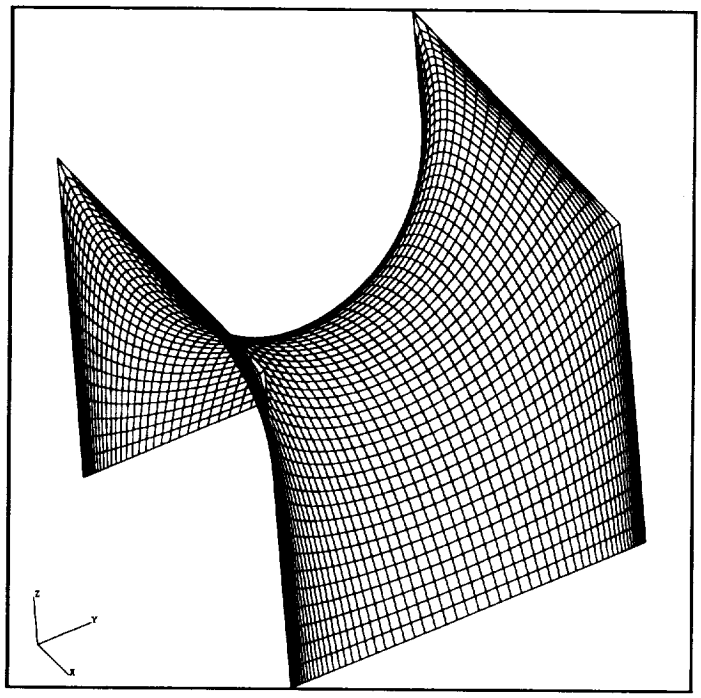


Figure 8: Minimal surface grid. Surface is a square Scherck surface.

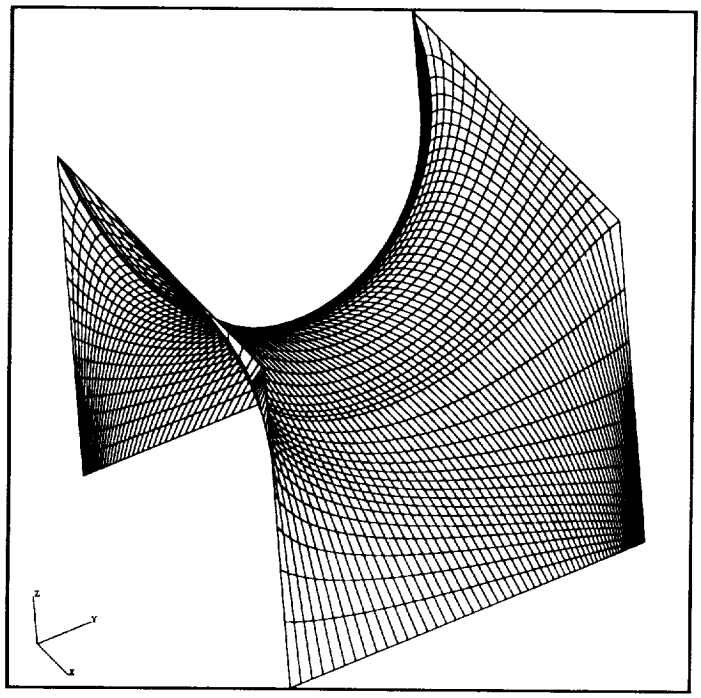


Figure 9: Minimal surface grid. Shape of surface is independent of the boundary grid point distribution.

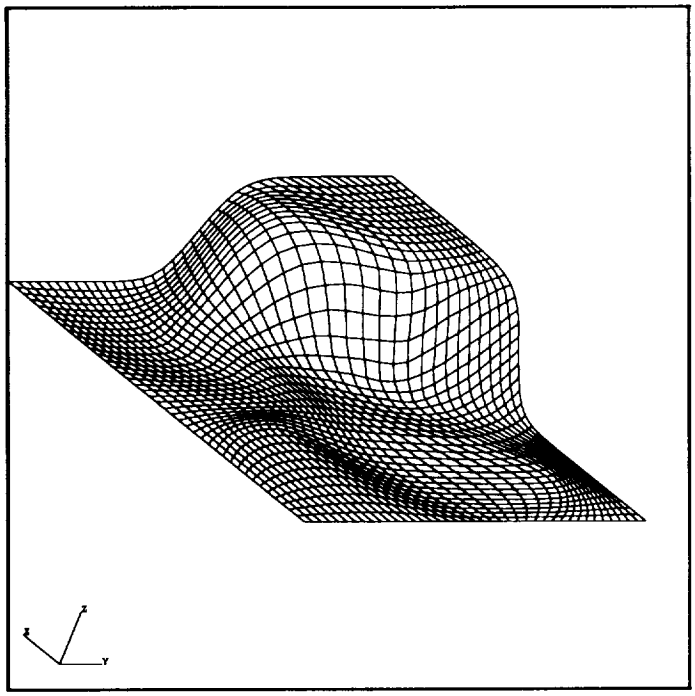


Figure 10: Irregularly distributed control point mesh on a smooth surface.

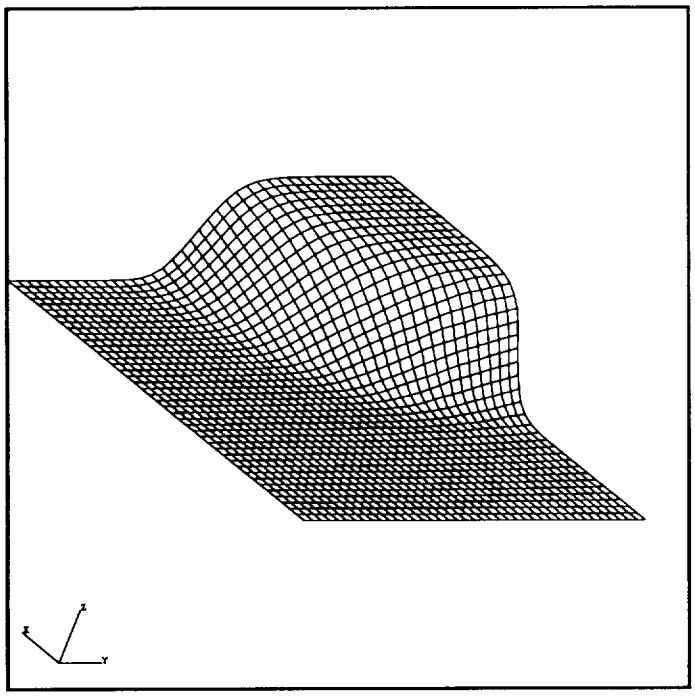


Figure 11: Elliptic grid on the surface. Grid is independent of the parametrization.

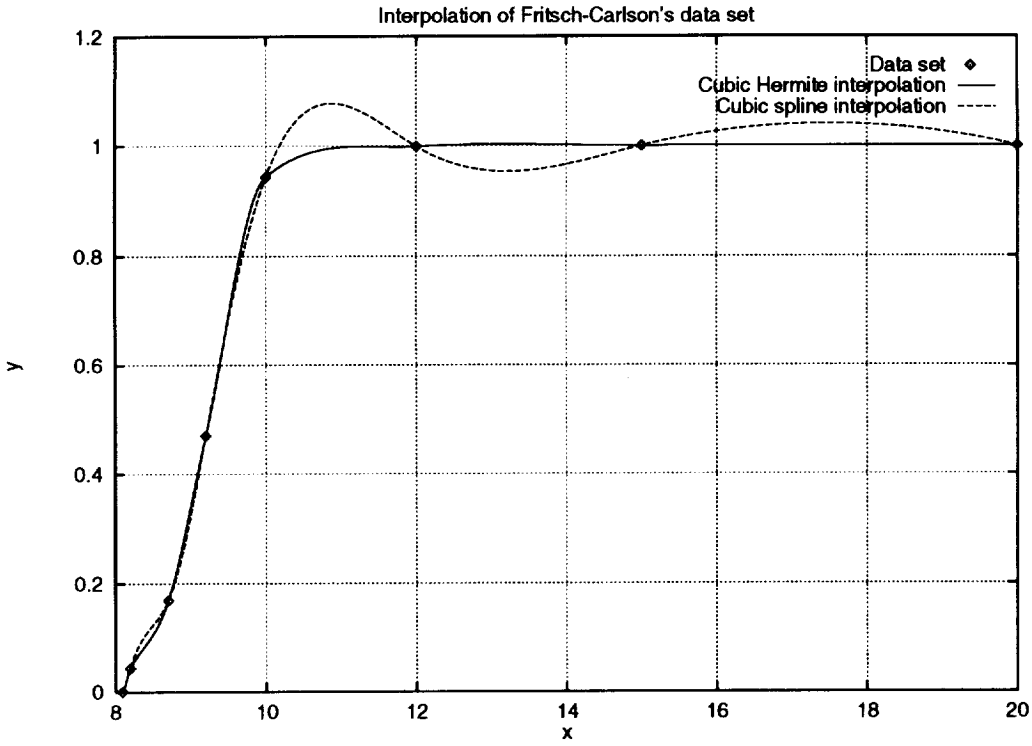



Figure 12: Comparison of cubic Hermite and cubic spline interpolation.

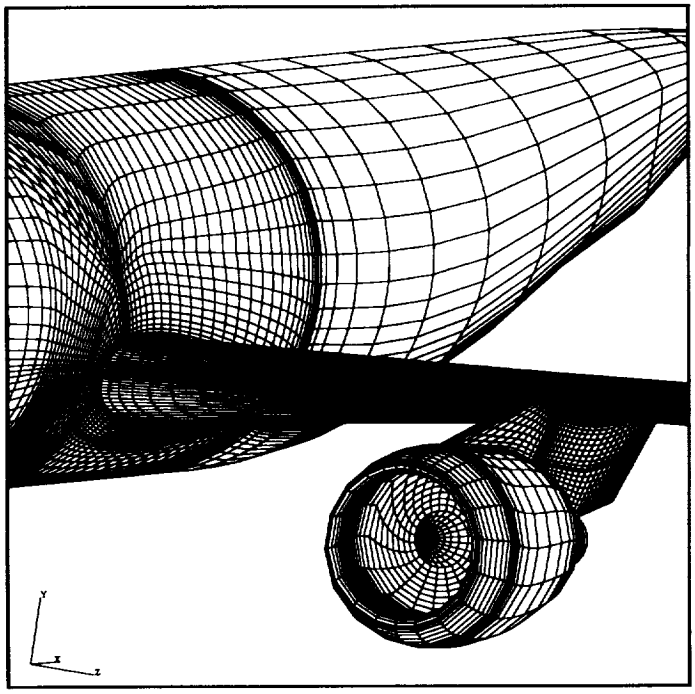


Figure 13: Surface grids of a wing-body-pylon-nacelle configuration.

N95-28761

# COOPERATIVE SOLUTIONS COUPLING A GEOMETRY ENGINE AND ADAPTIVE SOLVER CODES

Thomas P. Dickens
Aerodynamics Configuration Computing
The Boeing Company
Seattle, WA

## ABSTRACT

Follow-on work has progressed in using Aero Grid and Paneling System (AGPS[1,2]), a geometry and visualization system, as a dynamic real time geometry monitor, manipulator, and interrogator for other codes, as first published in 1992[3]. In particular, AGPS has been successfully coupled with adaptive flow solvers which iterate, refining the grid in areas of interest, and continuing on to a solution. With the coupling to the geometry engine, the new grids represent the actual geometry much more accurately since they are derived directly from the geometry and do not use refits to the first-cut grids. Additional work has been done with design runs where the geometric shape is modified to achieve a desired result. Various constraints are used to point the solution in a "reasonable" direction which also more closely satisfies the desired results.

Concepts and techniques are presented, as well as examples of sample case studies. Issues such as distributed operation of the cooperative codes versus running all codes locally and pre-calculation for performance are discussed. Future directions are considered which will build on these techniques in light of changing computer environments.

## INTRODUCTION

Current computing technology enables increasingly complex geometric configurations to be analyzed and allows analysis at a much finer level of detail. Flow solvers must work with the geometric definitions more closely to accommodate the level of accuracy desired. Rather than building more sophisticated geometry capabilities into the solvers, which is a duplication of the effort that went into the geometry code, a cooperative environment can be created which facilitates multiple codes working together. This environment will use the specialized capabilities and knowledge of each system, rather than investing the time and money to replicate capabilities in multiple codes plus the incurred problem of diverging capabilities once they are duplicated. The application of the code cooperation is needed in the geometry/solver combination due to the complexity and specialization of each of the two fields. This coupling of codes is especially beneficial in adaptive solvers, and when using solvers in a geometry-design mode.

In an aerodynamics design mode, a pressure solution is first generated for a geometric configuration. This pressure solution is then examined and modified by an engineer to represent a desired condition using a method of their choice. An area of the geometric configuration is selected for modification and may have associated constraints imposed. The solver then systematically perturbs the geometry while tracking the effects on the resulting solution. A new geometric configuration is calculated based on the results of the perturbations. A solution is then run for this new geometric configuration, which is then compared with the desired solution. This process is iterated as required.

It should be pointed out that, in this scenario, the geometric configuration is paneled, and then the resulting paneling is provided to the solver. The solver then perturbs the paneling, which represents the geometric configuration, to achieve the desired solution. The assumption with the paneling is that it is sufficiently refined to represent the actual surfaces within a desired tolerance. In these design runs, when the paneling is modified to represent new geometry, the original tolerances used to

generate the paneling may now be greater than desired to accurately represent the geometry. The solver itself does not understand the original geometry, nor does it know the original criteria which went into the paneling decisions. I have seen design cases which generated fairly sharp grooves and bulges in surfaces, as seen in Figure 1. If the original paneling was just within a desired criteria (chord-height tolerance with the geometry for example) changes to the design region can easily generate panels which, if tested back against the original geometry, are no longer within tolerance. This may introduce unwanted artifacts into the solution which degrade the usefulness of the achieved solution. Once the design run has achieved a target geometry which satisfies the desired pressure, the new geometric shape must be generated in the geometry tool (not a trivial task), a new paneling created, and a solution run on the new geometry to test the modified geometry.

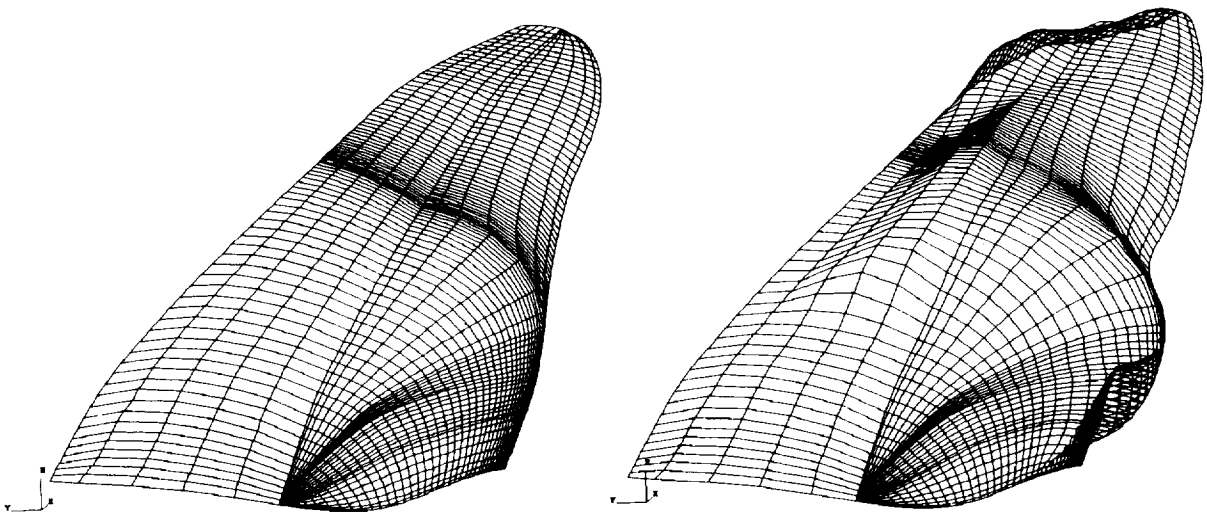


Figure 1. Before (left) and After (right) Panels of a Strut.

Another concern is the use of constraints within the solver. Localized constraints are used which can constrain the resulting grid to fit within a curvature tolerance, for example, or to impose a constraint to only allow movement in the positive surface-normal direction. Localized constraints are the first step in the process, but they do not have the necessary influence on the global properties of the configuration. Without more global constraints, local constraints can generate oscillations or other undesired artifacts in the resulting paneling which are out of the scope of the local constraints to "see".

The last step in the process is to create a new geometry loft which represents the modified panel data: the resulting output from the design-mode solver. The input panels were originally generated by discretizing the surface lofts within the geometry tool. The surface lofts are typically generated from a series of airfoils in the case of wing-like surfaces, or station curves for body-like surfaces. It should be noted that where two surfaces intersect, such as a wing-body intersection, the surface loft of the wing typically extends into the body. The extracted paneling is only for the exposed wing surface, the subrange area on the wing which is outboard of the wing-body intersection. The modified panel networks which are returned by the solver are used as templates to generate a new loft.

Several challenges occur here. There may be geometric shapes such as bumps and grooves in the new panels which are located between the original airfoils used for the original loft. Additional airfoils may be required for the new loft to sufficiently capture these shapes. But there are problems with using too many airfoils; high frequency noise can be introduced in the loft. Another problem is re-creating the parts of the loft that were not paneled, such as the part of the wing which is inside the body. The original loft data may be used, but if there are changes required in this region, the original data may not be useful. In this process you also wonder about all of the ripples in the new panels: are they necessary to achieve the desired solution, or are they artifacts of the process and not absolutely necessary? Once a new loft is generated, it is then paneled and processed again by the

solver with the hopes of matching the previously obtained results achieved from the design run.

## TERMINOLOGY OVERVIEW

To facilitate communication with a common understanding, some foundation definitions and concepts are outlined:

Geometric Definition.–Geometry is defined by two criteria: data, and functions which utilize the data. For example, a 3D surface is represented using a 2D parameter space. A function for evaluating the surface will return surface properties for a given parametric pair: D = F(S, T), where the returned data, D, is the data at the given S and T parametric location on the surface, and can consist of the 3D spatial coordinates, derivative information, surface normal, and other associated information. Higher dimensionality can be used to define geometry which would also be accessed by the given function. A surface can be constructed through an array of 3D data points to yield a set of data. For example, if the geometric form is a series of bicubic patches in 3D space, each patch will have 48 pieces of data to define the physical points at the four patch corners, plus the parametric derivatives with respect to S, T, and the ST cross derivatives. On top of this is the organization of the multiple surface patches to the parent surface. The surface evaluation routines are synchronized with the surface construction routines within a geometry tool. Without visibility of the methods and equations involved within the evaluation function, other codes can only approximate the interpretation of the geometry defined by the data. Additional complexity is added when the geometry tool builds upon the geometric definitions with techniques such as subrange and trimmed surfaces as well as procedural entities.

The key point here is that the data alone is not enough to accurately represent geometry. The defined evaluation process, or geometry engine, to work with the data is also required. Two different evaluation processes operating on the same data may not produce the same results. Thus, even if a solver has the capability to work with geometric data rather than just paneling representation of that data, unless the solver uses the same exact code as the geometry definition tool, the resulting geometry as known to the solver will be different than the geometry as defined in the geometry tool.

The Paneling Burden.–In the current computational fluid dynamics (CFD) process, the following tasks are performed: creating geometry, paneling the geometry, running the CFD code, and analyzing the solution. The particular CFD code used here is a full potential code that uses a locally refined rectangular grid which is generated internally by the code. In terms of flow time, the paneling process continues to be a bottleneck in the overall CFD process. Engineers currently can spend days paneling a typical wing-body-strut-nacelle configuration, and the resulting quality of the CFD solution is directly affected by the quality of the paneling. The engineer is burdened with the task of ensuring that the paneling is detailed enough for the problems being addressed to achieve meaningful results, yet not too dense to break the solver code or take too much time to run. Common techniques are used to help automate the paneling process. Such techniques are based on chord-height tolerance of the flat panel to the surface or other methods. The paneling process is not a bottleneck if the task can be automated, such as by using AGPS command files written to handle the topology of the configuration.[4]

Another concern is in iterative adaptive solutions, where emerging details in the solution will concentrate volume grids in areas of interest, such as shocks. The volume grids can generally be easily refined to detail these areas of interest. However, where the volume grid meets with the paneling, the flow code typically does a simple interpolation of the paneling to determine the panel/field intersection. It is at this panel/field intersection where additional knowledge of the actual geometry is needed. If the paneling in this region is not fine enough to accurately represent the geometry to achieve a reasonable solution, the task must be redone with a modified panel. This requires engineers to study solutions, to refine the paneling, and then rerun the codes. This is costly both in time and money. Of even greater risk is that the effect of the nonsufficient paneling is not realized and that an inaccurate solution is believed to be more accurate than it is. This is because the CFD code deals only with discretized geometry (paneling) and not with the original surface lofts. Again, the burden is placed on the engineers to understand the limitations of their paneling and to interpret the given solution with the appropriate level of accurateness. Of course a given paneling

may be intended to show general tendencies and is not intended to be used for finer details.

AGPS Overview.–While developed and used primarily for the preliminary design of aircraft, AGPS is used throughout The Boeing Company (Boeing) for a variety of geometric tasks. This Boeing proprietary program is a surface geometry system in which virtually any shape can be modeled. The underlying mathematics include cubic and quintic polynomials and rational B-splines for representing curves, surfaces, and solids. The interface consists of a structured programming language with over 160 geometry-related commands, along with a mouse-and-menu-driven interface. Higher-level command files can be constructed and used as a macro capability to do complex or repetitive tasks very simply. Collections of command files are used as high-level packages to allow users to accomplish complex tasks by following an interactive menu-driven session. Hundreds of existing command files are included in the AGPS release, which runs on a variety of machines including VAX, SGI, HP/Apollo, IBM, and Cray. AGPS has been coded to be machine and architecture independent, and utilizes a dynamically allocated object data structure.

TRANAIR Overview.[5]–A full-potential, solution-adaptive, rectangular grid code for predicting subsonic, transonic, and supersonic flows about arbitrary configurations, TRANAIR is used to analyze complex geometric configurations in transonic flow. A locally refinable rectangular grid is automatically constructed and is superimposed on the boundary geometry as described by networks of panels. Surface-fitted grid generation is not required. The nonlinear discrete system is solved using a preconditioned Krylov subspace method embedded in an inexact Newton method. The solution is obtained on a sequence of successively refined grids.

## GEOMETRY USE IN SOLVERS

There is an increasing trend to use solvers in a geometric design mode, in which the solver will be modifying the geometry definitions. To successfully accomplish this, greater attention must be placed on the geometry operations as driven by the solvers. There are two main directions to accomplish this: additional geometric capabilities can be coded into solvers, or the geometry tool and the solver can work together.

The geometric sophistication within solvers is limited. This is understandable and even expected; their domain of expertise is flow solving, and one does not expect to have solvers as sophisticated as dedicated geometry tools in the domain of geometry and geometric definitions and operations. Incorporating geometric sophistication within solvers makes the solvers much more complicated, and duplicates capabilities among various codes. Even if this route is taken, the geometric definitions will not be exact unless the same exact code used in the geometric tool is also used by the flow solver. The natural evolution for this path is to evolve the solver to also be your geometry tool, or to add solver capabilities to your geometry tool. Each of these fields require specialized expertise and generate extremely complex systems as they currently are. If a solver and geometry tool are merged into a single code, the resulting code could be too complex to manage.

A logical alternative approach is to dynamically couple the solver code with the geometry code, each working in a cooperative manner on the part of the problem which is its specialty. This approach has advantages and disadvantages. The primary advantage is the ability to refer back to the original geometric definitions. This is used for panel refinements as well as for moving the geometry and repaneling the changed area. By definition, in this process the paneling will define the geometry to the specified accuracy. Geometry changes will be transformed into new paneling which maintains the desired accuracy in its representation. With the geometry tool used in the design process, changes to the paneling can be applied to the loft process instead, which will then produce a modified loft. New paneling is then generated from this new loft. Once the design process achieves an acceptable solution, the loft which generated the paneling for that solution is already there. The problem of creating a new loft to represent the results of the design process is eliminated.

The disadvantages to this approach are the requirement of enabling the codes to cooperate together, as well as the performance overhead of the communications between the codes. Once the direction to enable codes to cooperate is taken, relatively minor modifications can be done to the

codes involved to accomplish this task. This leaves the performance penalty in the communications between the codes as the major disadvantage.

Compared with the alternatives of either working in the current paradigm or incorporating the modifications necessary to put sophisticated geometric capabilities into a solver (or the reverse), I contend that the advantages of coupling these codes strongly outweigh the disadvantages.

## COUPLING MULTIPLE CODES

Within Boeing, work has been done in this direction. I have added the capabilities to AGPS to allow various cooperation methods between codes. This cooperation can be set up in two different schemes, based on which programming unit (AGPS or another code) in the system is in control.

External control, or what we call the *master* type of connection, allows another program to connect to AGPS and directly interface with the AGPS command line input stream. The external process also will receive all of the output from AGPS (Figure 2). In effect, the external master program appears to AGPS to be a user interacting with the normal input and output capabilities of AGPS. This capitalizes on a very successful feature of AGPS: the command-line interface. The AGPS development team has recognized the importance of offering almost all of the capabilities within AGPS in a command-line interface as well as a graphical user interface (GUI) point-and-click interface. This feature allows repetitive tasks to be easily automated and repeated with little or no user interaction required. Routines which initiate and terminate the controlled AGPS connection, as well as send and receive data between AGPS and the master program, are provided by the AGPS development team to allow this capability to be easily incorporated into the I/O system of a master program. This feature allows an external program to easily assume the role of the AGPS user and to have AGPS accomplish geometric tasks for the program.

There is a second level of master program access into AGPS. The external program can directly invoke a set of AGPS internal subroutines which handle the geometry creation, modifications, and interrogation within AGPS. The direct routine access capabilities are generally not used by external programs; the command-line interface is the most widely used method to control AGPS via an external program.

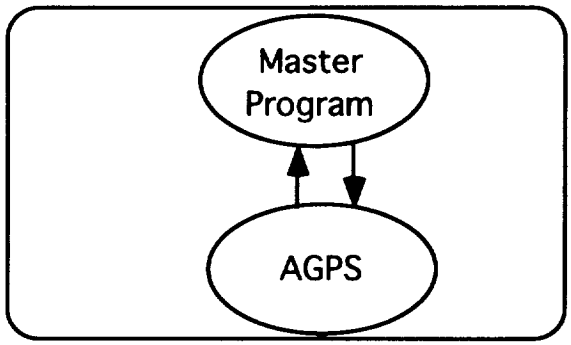



Figure 2. Master Program Using AGPS Capabilities.

The second scheme is to have AGPS in control. Up to nine external processes, referred to as *slave* processes, can be dynamically invoked and connected to AGPS (Figure 3). A slave process is spawned from AGPS, with AGPS remapping the standard-in and standard-out streams of the slave process into pipes which are held by AGPS. This way the slave process does not need to be modified in any way to allow it to be used as an AGPS slave process. This was a very important concept for the general usefulness of the slave capability. If two communicating programs need to jointly agree on an interface specification and protocol and need to be maintained with the knowledge of the other, the interdependence between the codes may be unworkable. In addition, the coupling of a different pair of codes will require modifications to both of them to allow them to communicate. With the approach of mapping the standard-in and standard-out streams of the invoked process, the invoked

process does not require any modifications. Any process or program available on the system can be utilized as a slave process. This allows AGPS to have at its command an endless supply of dynamic capabilities. This is used in production to provide added capability to AGPS, and to prototype new AGPS capabilities without requiring modifications to AGPS.

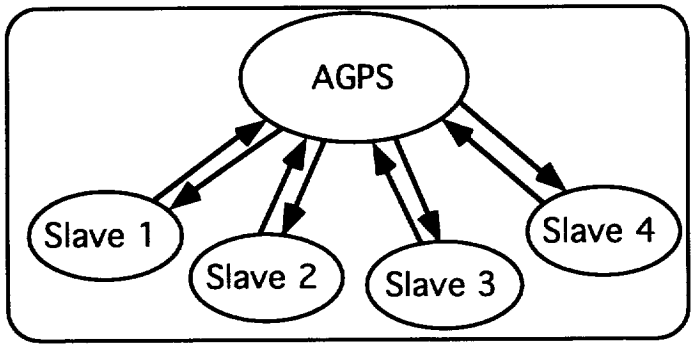



Figure 3. AGPS Using Multiple Slave Programs.

Tradeoffs.–With the techniques of coupling two or more codes to cooperatively accomplish a task, there are costs and compromises involved. On the positive side, the savings in development time to incorporate major new capabilities are significantly reduced from months or years into a few hours or days to access the capabilities through existing external codes. This provides the new capabilities almost immediately when needed. Specialized expertise is not required to develop the capabilities from scratch, and the capabilities are known to be mature and robust. On the negative side of code coupling are performance issues. With two or more processes communicating, there will be an overhead involved with physically running the programs. More important is the limited bandwidth for communication between the processes and the sequential nature of tasks. These issues can be addressed and minimized while still maintaining the general-purpose nature of the solution.

However, attaining the same performance as compared to a dedicated program is not achievable. My feeling is that if the overall solution is no more than an order of magnitude slower, the performance penalty is acceptable, at least for initial proof of concept purposes. The bottleneck of the communication overhead can be minimized through the use of files to communicate large quantities of information, using process-to-process communication primarily for control. Another technique is to partition the task in ways that minimize the communication required and allow for larger amounts of work to be accomplished for a given set of data. This can be combined with the concern of sequential work by front-loading the known work to be done with an external system to allow the external system to have as much of the work completed when the master system requires the results. As demonstrated below, external tasks can be front-loaded and also distributed among multiple machines to achieve more responsive turnaround.

## CASE STUDY: TRANAIR AND AGPS

As a case study of these techniques, we will consider the use of TRANAIR in a design mode which uses AGPS as a geometry engine.

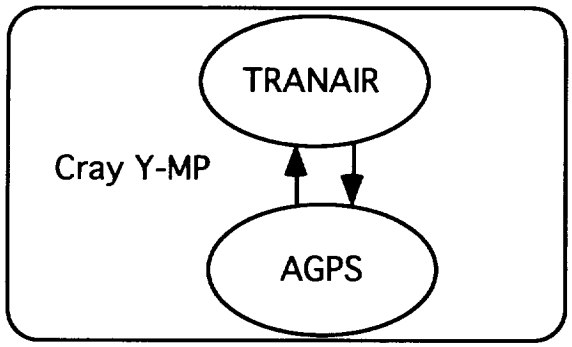



Figure 4. Coupling of TRANAIR and AGPS on the Cray.

The first implementation of the TRANAIR/AGPS coupling used both codes running simultaneously on the Cray Y-MP (Figure 4). It should be noted that TRANAIR has been optimized for the Cray architecture, while AGPS has not. Add to this the interpreted command-line nature of AGPS which results in a very generalized geometry system, but a system which is much slower than dedicated code to accomplish specific tasks. Code was added to TRANAIR to launch AGPS, and to communicate the desired geometry tasks to AGPS. The algorithms involved to solve the geometry tasks were implemented in AGPS command files which allowed easy development and modifications to the algorithms without requiring subsequent changes to either of the two systems. TRANAIR requested AGPS to perform a geometry task and then waited for AGPS to finish the task. AGPS communicated the result to TRANAIR via a data file, which TRANAIR read. For a sample design case using 40 design points, 80 solutions were required of AGPS for each iteration. Needless to say, this was slower than implementing similar capabilities directly into TRANAIR. Each iteration using AGPS took about a minute to complete, and then TRANAIR took a few seconds to digest the information. A similar implementation coded solely into TRANAIR took a few seconds per iteration. These results must be weighed with the fact that the AGPS solution was configured to run within a couple of days while a TRANAIR version modified with similar capabilities took weeks (flow time) to be available, and the capabilities were job-specific and hard-coded into the code.

Much was learned from these experiments. A small change in TRANAIR was made which launched the 80 AGPS jobs as soon as TRANAIR knew what those jobs consisted of--which was about 10 minutes before TRANAIR required the results. This gave AGPS some lead time to get a head start on the task. It was also noted that the initial conditions were known before the run was even started, so the first iteration results could be pre-calculated by AGPS before the TRANAIR run was initiated. During the process we were also able to gather metrics on the running of TRANAIR, which allowed bottlenecks within the TRANAIR design-mode code to be discovered and optimized.

The next logical step in the experiment was to distribute the AGPS tasks to a local workstation (Figure 5). Since AGPS is not optimized for the Cray architecture, wall-time performance on workstation class machines was comparable to the shared Cray performance. Using a handful of UNIX scripts and files for communications between the two machines, running TRANAIR on the Cray and the AGPS geometry jobs on a workstation gave comparable results to running both codes on the Cray. This has the added benefit of off loading the geometry task from the Cray which allowed our Cray resources to be focused on the solver task. Even with the 10 minute lead time, AGPS still could not keep up with TRANAIR's need for results when it required them.

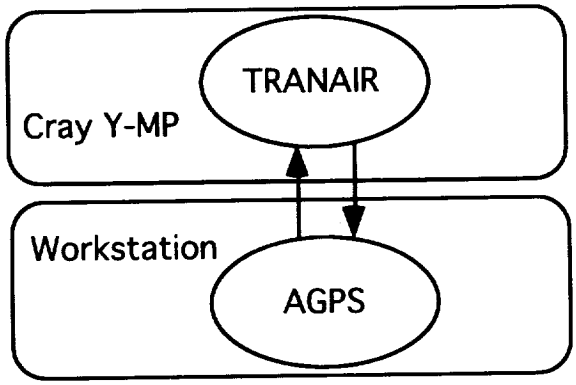



Figure 5. TRANAIR and AGPS Coupled Across Different Machines.

We then looked at distributing the geometry task across multiple workstations. This was initially done using UNIX scripts to launch remote AGPS sessions on predetermined machines and to assign a predetermined set of jobs to each machine (Figure 6). A central machine was used to collect the results and communicate them to the Cray. When this scheme worked, it worked well. We were able to feed the results to the Cray fast enough to keep up with its demand; however, this scheme is quite fragile in practice. With the list of machines predetermined, and the jobs on these machines predetermined, if one machine was either unavailable or heavily loaded during the run, the job would fail due to an incomplete set of results. Some system-level glitches were discovered, such as the inability to spawn remote jobs on the Cray with a large number of files opened. We finally tracked this glitch into the system kernel on the Cray as a problem with the rsh command.

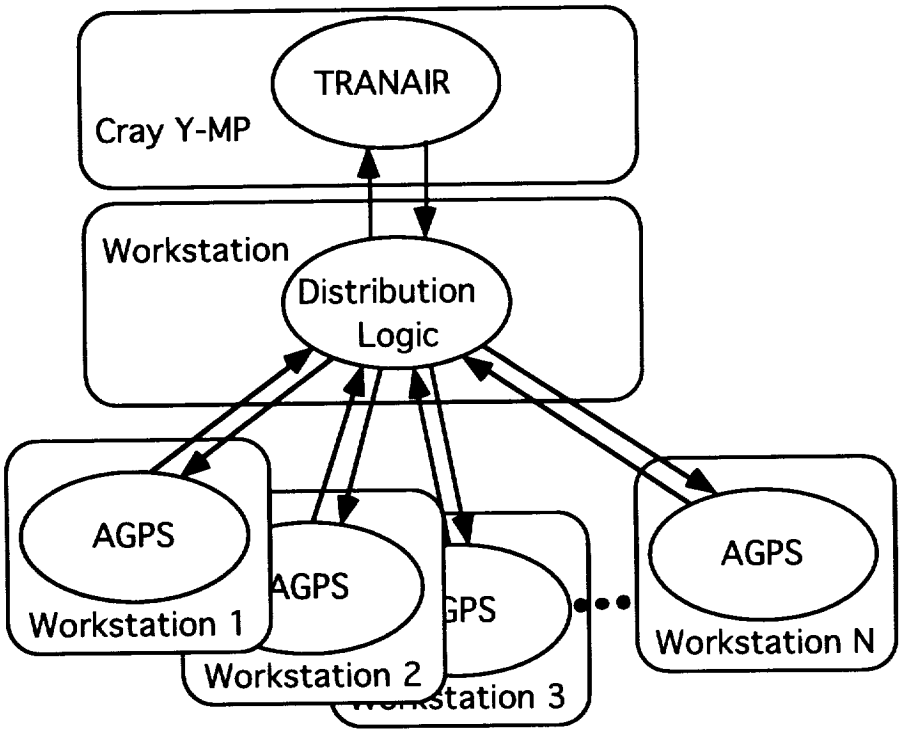



Figure 6. TRANAIR with Multiple AGPS Coupled Across Many Machines.

We are currently working to refine these techniques and to offer them to the Boeing user community in an easy-to-use manner.

## CONCERNS AND FUTURE WORK

During this effort we discovered various roadblocks and have identified areas needing enhancements.

Receiving Complete Files.–Three problems may occur in the transmission of files between the various machines. One problem is the corruption of a file. This is rather rare due to the robustness and error checking built into ftp and rcp. A second problem is the non-arrival of a file. This does happen and can be caused by network problems, file-space problems, or code failure on the sending machine. A third problem occurs rather commonly; the use of a file before the entire file is received. There is no mechanism built into UNIX to make a file unavailable until it is complete. Various techniques can be used to address this, such as including the checksum of the file as a field in the name of the file. The receiving program will not open or access the file until the checksum generated from the file matches the checksum in the filename. This has the added benefit of also checking for corrupt files. Another method is for the sending machine to send the file with a temporary filename, then, after transmission, to remotely change the name of the file. This should work in theory, but subtle timing through the network can cause failure. The transmission can be completed and then the file name changed, while actually the file is not completely available on the remote system.

Working with Dynamic Machine/Network Loading.–At this time, work is being done on a C program which will dynamically determine the health of a given set of machines on the network and issue geometry tasks to be solved with AGPS jobs running on the subset of available machines. These tasks will be monitored for completion and also for performance. Remaining jobs will be allocated in a logical manner to ensure completion of the entire set of tasks as quickly as possible as well as the ordering of the results received. Tasks may be reissued on top performers if the assigned machine is not responding in a timely manner, or if the resulting files are not complete. This approach will offer many improvements. Most failures in the system can be recovered from during a run, including a node being unavailable, a node being too heavily loaded with other tasks, network failures, and file corruption. Problems can still occur if the central machine coordinating the distributed effort goes down, loses its network connection, or is too heavily loaded. Preliminary testing of this capability looks very promising.

## SUMMARY

Benefits of using AGPS for geometry tasks.–It has been demonstrated that there are a variety of benefits in coupling two or more specialized programs together in a cooperative manner. These benefits must be weighed with the costs incurred to determine the appropriateness of this technique for a given environment and problem. The costs include both development costs and run-time costs. There will be a development effort involved to set up the programs for cooperative communication. Doing the development changes in a general-purpose manner, and implementing the main capability in a key central program which does not dictate changes in other programs, is highly favorable. The run-time costs include the overhead of running multiple codes and running codes not optimized for a particular task. This results in both higher CPU usage costs as well as longer wall-time turnaround. These costs can be minimized by partitioning the problem. Benefits include the rapid availability of new capabilities, use of capabilities without requiring extensive code development, gaining access to specialized code without duplicating or reproducing the code, and immediate access to robust, mature code.

Easy-to-change tasks (command-file driven and dynamically accessed capabilities).–By using the AGPS command language to accomplish the geometry tasks, algorithmic changes to the command files can be incorporated without requiring changes to any compiled source code. This allows very rapid turnaround of modifications to the geometric algorithms. In addition to production work, this facilitates rapid prototyping of new capabilities without requiring the system to be modified and rebuilt. The expertise needed to work in this environment does not require an expert with the internals of either code.

Work is done on the actual geometry, in the defining system of the geometry.–Refinements made at the request of the solver can be made to the actual geometry, rather than using extracted panels and

approximating the defining geometry. As geometric changes are made, effects to the surfaces can be realized and used. Dynamic refinements to the paneling is possible to accommodate areas of interest as the solution evolves.

AGPS has the tools to work with global constraints/conditions.–Using the capabilities in a mature geometry system provides a much richer set of offerings than is available as add-on capabilities in a solver. These offerings are also proven to be reliable due to widespread use for other applications over time, while new code added into solvers will not be mature or widely tested. Many years of work have gone into the geometry system, which results in a system which is fine tuned to work with geometry, both on a local scale and in a global scale. This global understanding of the geometry, and the ability to work with the geometry in this manner, allows better constraints to be imposed on the solution which will lead to more usable solutions.

Changing loft definitions in the design mode.–By using AGPS to dynamically modify the loft definition during a design run, the problem of creating a new loft based on a set of modified panel networks is eliminated. The cost to run a solution for the new loft is also eliminated. The result of the design run is a new loft rather than modified panel networks.

Generality–The techniques presented here are appropriate to be applied in a general manner to numerous other applications and should not be limited to CFD solvers and geometry codes. Candidate applications are those where two or more specialized codes working together can yield a cooperative benefit.

## ACKNOWLEDGMENTS

I would like to thank Arvel Gentry for his shared vision and his long hours using and evolving these techniques. I also appreciate the cooperation and knowledge from the TRANAIR development group and the other members of the AGPS development group.

## REFERENCES

1. D. K. Snepp and R. C. Pomeroy, "A Geometry System for Aerodynamic Design," AIAA/AHS/ASEE Aircraft Design, Systems, and Operations Meeting, AIAA-87-2902, September 14-16, 1987.

2. W. K. Capron and K. L Smit, "Advanced Aerodynamic Application of an Interactive Geometry and Visualization System," 29th Aerospace Sciences Meeting, AIAA-91-0800, January 7-10, 1991.

3. T. P. Dickens, "Technique for Using a Geometry and Visualization System to Monitor and Manipulate Information in Other Codes," Software Systems for Surface Modeling and Grid Generation, NASA Conference Publication 3143, April 28-30, 1992.

4. A. E. Gentry, "Requirements for a Geometry Programming language for CFD Applications, Software Systems for Surface Modeling and Grid Generation," NASA Conference Publication 3143, April 28-30, 1992.

5. F. T. Johnson et al., "TRANAIR: A Full-Potential, Solution-Adaptive, Rectangular Grid Code for Predicting Subsonic, Transonic, and Supersonic Flows About Arbitrary Configurations," Theory Document, NASA Contractor Report 4348, December 1992.

N95- 28762

# AUTOMATION OF THREE-DIMENSIONAL STRUCTURED MESH GENERATION FOR TURBOMACHINERY BLADE PASSAGES

Edward P. Ascoli and George H. Prueger
Rockwell International/Rocketdyne Division
6633 Canoga Avenue, P.O. Box 7922
Canoga Park, CA 91309-7922

## ABSTRACT

Hybrid tools have been developed which greatly reduce the time required to generate three-dimensional structured CFD meshes for turbomachinery blade passages. RAGGS, an existing, Rockwell proprietary, general purpose mesh generation and visualization system provides the starting point and framework for tool development. Utilities which manipulate and interface with RAGGS tools have been developed to 1) facilitate blade geometry inputs from point or CAD representations, 2) automate auxiliary surface creation and 3) streamline and automate edge, surface and subsequent volume mesh generation from minimal inputs. The emphasis of this approach has been to maintain all the functionality of the general purpose mesh generator while simultaneously eliminating the bulk of the repetitive and tedious manual steps in the mesh generation process. Using this approach, mesh generation cycle times have been reduced from the order of days down to the order of hours.

## INTRODUCTION

Over the last decade, advances in both computer technology and numerical algorithms have made Computational Fluid Dynamics (CFD) a viable analysis tool for engineering design. In particular, aerospace companies have employed CFD in the design of turbomachinery components, combustion devices as well as hypersonic propulsion systems. The major limitation to CFD as a design tool has been the length of the CFD analysis process. When modeling complex geometries the CFD analysis process is frequently too time intensive to be used early in the design cycle. Unfortunately, the later that CFD is used in the design cycle the less likely it is to have an impact on the final design.

Geometrical complexity impacts the CFD analysis process at each step. In the preprocessing step, the time required for model creation and mesh generation increases dramatically with increasing geometrical complexity. CFD meshes for complex geometries generally require more nodes or elements and thus the flow solution phase of the CFD process increases in time. Finally, visualization and data reduction in the postprocessing phase require more time and thought for complex geometries. On average, for complex geometries, the preprocessing phase of the analysis requires the greatest amount of time. In many cases preprocessing may require much more time than either flow solution or postprocessing combined. The primary focus of this work is on the preprocessing step of the CFD analysis cycle.

Preprocessing logically subdivides into model creation and mesh generation. The geometry may only exist in the form of diagrams or sets of points defining lines, curves or surfaces. Tools are then required to convert this data form into a model upon which a mesh may be constructed. In this case, a model may be created using any of a variety of CAD tools such as CATIA or Pro/ENGINEER. Mesh generation tools would then be required to import the CAD geometry. Alternately, mesh generation packages exist which have CAD or CAD-like capability.

A CAD model may have already been created to serve some other design purpose. Unfortunately, a CAD model suitable for rapid prototyping or automated machining etc. may be different than a model of the same geometry which is suitable for mesh generation. Extraneous surfaces may have to be removed while additional (auxiliary) surfaces may have to be created to extend or partition a flow region.

Once a suitable model has been obtained, mesh generation is performed. Although fully unstructured mesh CFD technology has been rapidly evolving, it has not reached the maturity to displace existing multiblock structured mesh CFD technology. This work focuses on structured mesh generation. Structured mesh generation requires up front planning to decide the general topological layout and blocking, as well as node number and distribution. In general, three-dimensional structured meshing proceeds serially from edge meshing to surface meshing and then to

volume meshing. Four edges determine the boundary of a non-degenerate surface while six surfaces determine the boundary of a non-degenerate volume.

The goal of this effort has been to reduce the time spent in the preprocessing phase of the CFD analysis cycle for a commercially important class of complex geometries. To this end 1) a relevant class of geometries was selected, 2) the primary grid generation tool was identified, 3) key geometry model input and generation issues were identified, 4) key flow solution requirements were identified and 5) tools were developed which automated and expedited the preprocessing phase of the analysis cycle.

## GEOMETRY CLASS

A relevant class of complex geometries, namely blade passages, was identified. Blade passages are elements in a wide variety of applications ranging from rocket engine turbopumps (ref. 1) to air compressors. The primary deterrent to using full three-dimensional CFD in the initial design of rocket engine turbopumps is the time required to generate the appropriate geometry model and corresponding mesh. In this effort, three types of blade passages were considered: axial, radial and mixed. Axial flow devices have inflow and outflow primarily in the axial direction. Radial flow devices have axial inflow and radial outflow. Mixed flow devices have inflow in the axial direction and outflow at a flow angle between axial and radial. The three configurations are shown schematically in figure 1. Figures 2 through 4 are turbopump components with either axial or radial flow configurations.

## PRIMARY GRID GENERATION SYSTEM

Rockwell Automated Grid Generation System (RAGGS, (ref. 2)), a general purpose mesh generation and visualization system was selected as the primary grid generation tool. RAGGS is a Rockwell proprietary code which was developed at North American Aircraft Division of Rockwell International. In interactive mode, RAGGS is a highly graphical tool with an extensive user interface. Geometry models as well as edge, surface and volume meshes are easily manipulated and visualized via menus and mouse control. Edge, surface and volume meshing may be performed in a highly interactive and visual manner in this mode as well. Of critical importance to the current effort is the fact that nearly every primary gridding tool available in interactive mode has a nongraphical, "batch" counterpart. For instance, a simple nongraphical tool exists to create an edge mesh on a specified surface. Similarly, nongraphical tools exist for surface and volume meshing. These tools are extremely accessible and flexible. The key point is that these tools may be combined to form macros. Since RAGGS runs in a UNIX workstation environment, C-shell and Bourne shell macros were written calling various tools. In this manner the grid generation process was expedited and automated while at the same time key intermediates were easily visualized and optimized in interactive mode.

## GEOMETRY INPUT AND GENERATION

Initial geometry data for blade passage configurations at Rocketdyne is available in one of two forms: point data or CAD geometry files. Generally, point data is available earlier in the design cycle. The format for point data is two arrays of points, one for each side of the blade. Together the two arrays may combine to form the two sides of a single blade or the two arrays may represent the bounding blade surfaces of a passage. The arrays are assumed to be N x M arrays (M curves with N points per curve) which span the leading edge to trailing edge, otherwise points are interpolated as appropriate. From the blade surface data and a few geometrical inputs (e.g., the number of blade passages) all other geometrical features can be constructed.

Since CAD design at Rocketdyne may be performed on a variety of CAD systems, CAD geometry files exist in any of the corresponding CAD formats. All of the CAD systems at Rocketdyne have translators enabling conversion to IGES (International Graphics Exchange Standard) format. RAGGS is able to input IGES geometry files and the CAD to IGES to RAGGS path has been tested for CATIA, Pro/ENGINEER, PATRAN, and UNIGRAPHICS. CAD geometry files obtained at this stage in the design cycle usually contain a large amount of superfluous detail (i.e. details of solid portions of the hardware item when flow passages are of primary interest, figure 5 ). Often the geometry may not contain a complete contiguous blade passage (i.e., the furnished geometry may consist of a "pie" or sector cut of the full geometry, figure 6). For this effort, when CAD data rather than point data was provided it was assumed that the format was IGES and that two complete blade surfaces could be

constructed from the IGES data provided. Again, the two surfaces could combine to form either a single blade or the bounding surfaces of a passage.

## FLOW SOLUTION CONSIDERATIONS

Simulation of the flow domain requires that the blade passage be extended both upstream and downstream. The length of the upstream and downstream extensions is an input provided by the analyst performing the flow simulation. Although the geometry details of the upstream and downstream extensions are flexible, the side boundary surface grids must be periodic since the single blade passage along with the extensions represents one element in a device with repeated elements in rotational symmetry.

The flow algorithm and physical modeling also dictate a host of grid features including wall clustering, cell aspect ratio, cell skewness and overall grid quality.

## TOOL DEVELOPMENT

The preprocessing phase of the analysis cycle was divided into the following steps: 1) blade surface input, 2) full geometry creation, 3) edge and surface meshing, 4) volume meshing and 5) mesh manipulation and quality checking. Corresponding to each step, a module was created.

The blade surface module converted either IGES blade geometry data or blade point data into a form which could be directly used by RAGGS. The output of the blade surface module could be read directly into RAGGS in interactive mode and examined for consistency and correctness.

The geometry module read user geometry input (e.g., upstream and downstream extension lengths and geometry details) as well as the blade surface geometry as output from the blade surface module. The geometry module used the inner and outer edges of the blade surface to construct the hub and shroud profiles. The blade leading and trailing edges were used to construct the upstream and downstream extensions respectively. The blade surfaces could be given as either the two sides of a single complete blade or as two sides of the blade passage. If the surfaces were given as a single complete blade the logic was in place to generate a blade passage. Based on geometrical input, two auxiliary surfaces were created, one spanned the entrance of the blade passage from leading edge to leading edge while the second spanned the exit of the blade passage from trailing edge to trailing edge. These two auxiliary surfaces partitioned the total geometry into upstream extension, blade passage and downstream extension. Partitioning the geometry in this way provided greater control over the three volume grids that were subsequently generated. In particular, these auxiliary surfaces were used to minimize mesh skewness at the blade leading and trailing edges.

The geometry module centered about FORTRAN programs written to create the new geometry elements. These FORTRAN programs along with existing batch RAGGS tools were linked via C-shell scripts. The output of the geometry module was a complete consistent geometry containing all features necessary to make the final mesh. This geometry could be read into RAGGS in interactive mode and examined.

The edge and surface meshing module read point distribution and clustering information from an input deck and automatically generated the appropriate edge and surface meshes. Initial surface meshes were generated with RAGGS transfinite mesh generation tools. For surfaces which were anticipated to contain excessively skewed cells, elliptic refinement was automatically performed. Each surface mesh was tested for negative area cells. If negative area cells were encountered, elliptic refinement was performed automatically. The output of the edge and surface meshing module was a set of files containing final edge and surface meshes. These files were easily displayed and modified in interactive RAGGS mode.

The volume mesh module read surface meshes from the edge and surface meshing module and produced transfinite volume grids. Each volume grid was tested for negative cell volumes. If a negative cell volume was encountered elliptic refinement was performed automatically. The final volume mesh was examined in RAGGS interactive mode.

The last module performed a host of miscellaneous manipulations and quality checks on volume meshes. Minimum, maximum, average and standard deviation were calculated for quantities like aspect ratio, skewness and cell volume. Grid index manipulations could be performed as needed. Thus the I index could be switched with the K index, etc. Two or more grids could be combined into a single grid. Since the blade passage grid was actually generated in three blocks corresponding to the inflow extension, the blade passage and the outflow extension, if the analyst preferred a single block, the three blocks were combined into one block with this tool. Mesh lines could be added (interpolated) or removed using the mesh manipulation tool. Finally, the mesh could be output in any of a variety of formats required by existing flow solvers.

## RESULTS

Figure 7 shows the three-dimensional mesh generated for the axial configuration modeling a three-dimensional turbine nozzle. The blade surface for this configuration was available both as blade point data as well as CAD IGES data. Both formats were meshed using the tools developed under this effort. Beginning with either blade point data or CAD IGES geometry data, the start to finish meshing time was less than five hours using the automated tools. The five hours roughly divides into three hours for geometry creation and finalization and less than two hours for actual mesh generation. The total of five hours compares to 25 hours for "manual" meshing from an existing CAD IGES geometry and 30 hours for "manual" meshing beginning from blade point data. When the geometry is available as blade point data, manual refers to, first, employing the services of a CAD expert to create the base geometry and generate the necessary auxiliary surfaces to partition the flow passage and, second, using RAGGS in interactive graphical mode to generate the mesh step by step. When the geometry is available as a CAD IGES file the first manual step is simplified since the CAD expert need not generate the base geometry.

The axial flow devices shown in figures 8 and 9 are both inducers. Blade point data was provided in each case. Start to finish mesh generation using the automated tools required between five and six hours for each configuration. This time compares to roughly 35 hours for "manual" (RAGGS and CAD expert) mesh generation. Note that it took two hours to produce a subsequent modified mesh for the configuration shown in figure 9. This modified mesh employed the same geometry as shown in the figure, but varied point distribution and clustering.

Figure 10 shows an impeller. Blade data for this radial flow device was provided in the form of a CAD IGES file. Time estimates for "manual" and "automated" generation of this configuration are nearly the same as those given above for the inducer configurations.

## FUTURE WORK

Future work will focus on including a variety of additional geometrical features. The overall approach can be modified to include partial or splitter blades in the blade passage. Tools exist to handle tip clearances between the blade and the shroud but these tools have not been incorporated into the current framework. Subsequent work will improve these tools and incorporated them into the overall system. Fillets at the intersection of blade surface with the hub or shroud could eventually be taken into account. Currently, blade surface IGES files which are based on trimmed surfaces are excluded. Future work will eliminate this limitation. Finally, this work has focused on rotating machinery blade passages. The approach that has been taken is sufficiently general that it can be applied to nearly any geometric class to produce template type capability. Future work will focus on other geometric classes to produce preprocessing templates analogous to the one presented in this paper.

## REFERENCES


1. AIAA-92-3400, "Application of a Computational Engineering System to Analysis of a Rocket Engine Pump Impeller," Steven L. Barson, Edward P. Ascoli, Daniel C. Chan, Michele E. DeCroix, and Munir M. Sindir, 28th Joint Propulsion Conference and Exhibit, July 6-8, 1992, Nashville, TN.

2. AIAA 94-0206, "RAGGS: Rockwell Automated Grid Generation System," Chung-Jin Woan, Willard C. Clever and Clement K. Tam, 32nd Aerospace Science Meeting, Jan. 10-13, 1994, Reno Nevada.

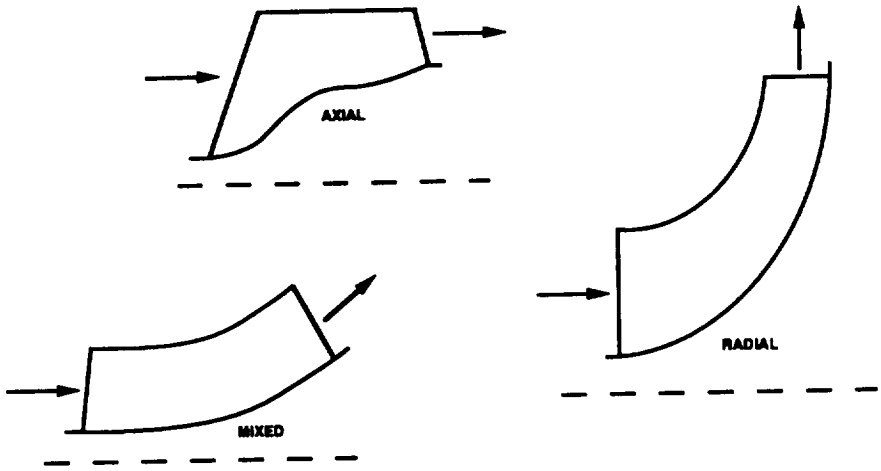



Figure 1. –Schematic of axial, radial and mixed blade passages.

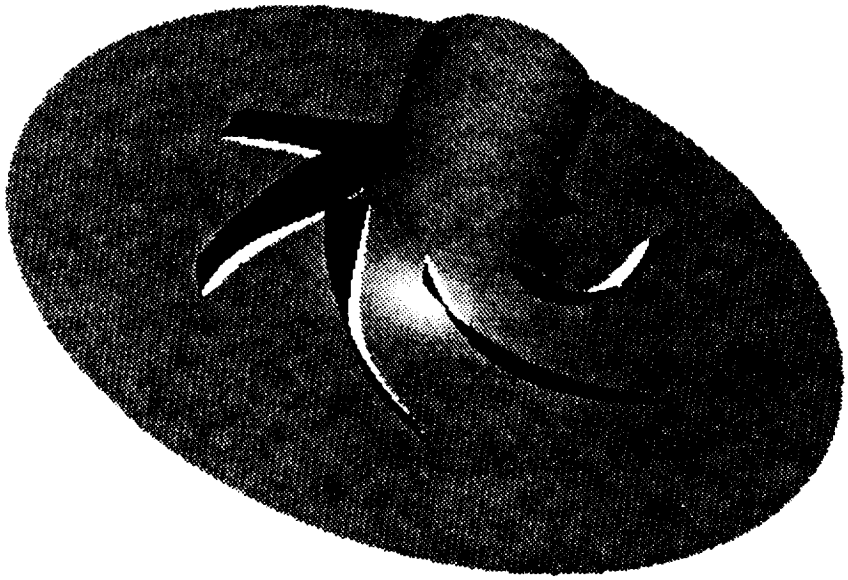


Figure 2.–Radial flow device (impeller).

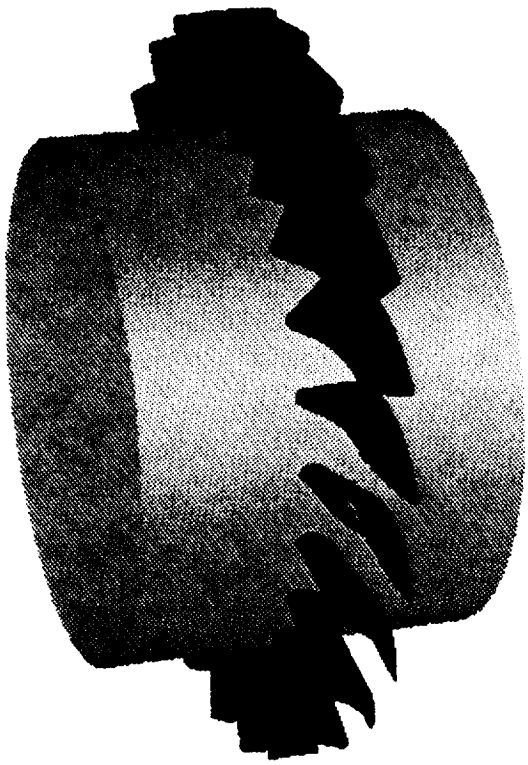


Figure 3.–Axial flow device (three-dimensional turbine nozzle)

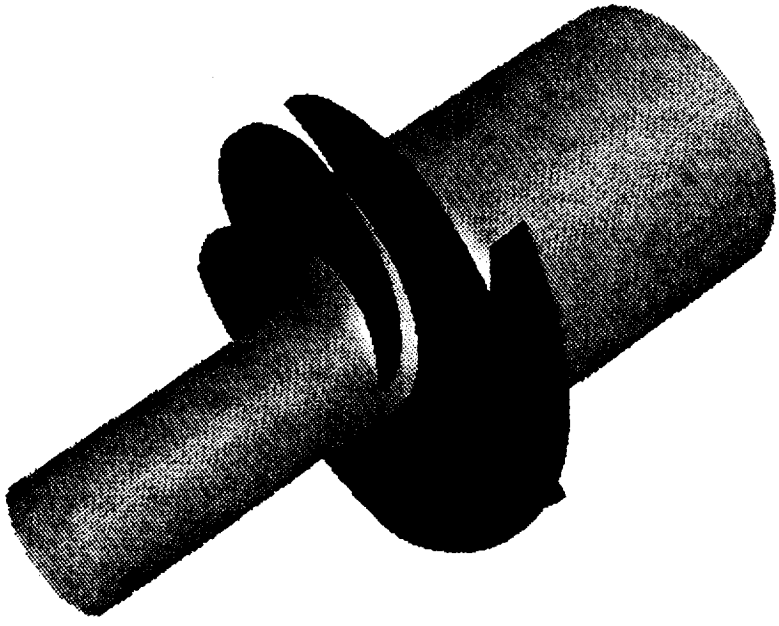


Figure 4.–Axial flow device (inducer)

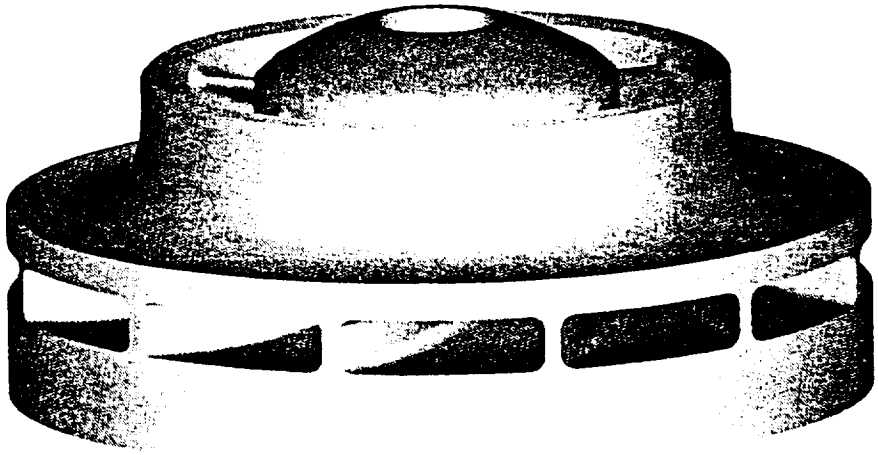


Figure 5.–CAD geometry representation of an impeller showing full 360° display and superfluous hardware detail.

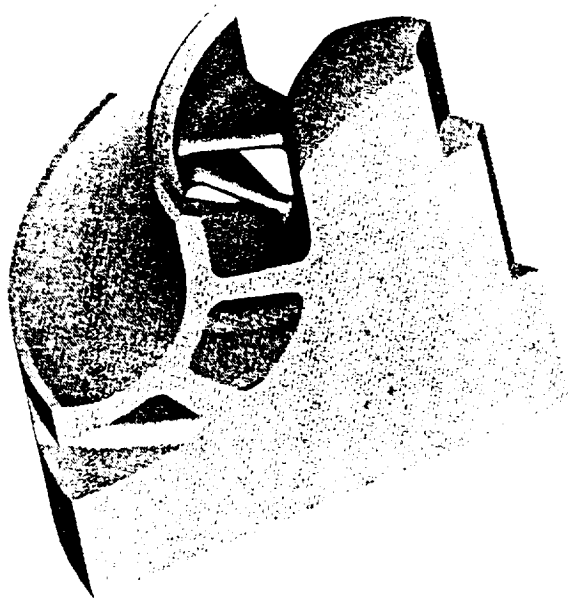


Figure 6.–A "pie-slice" or sector cut of the geometry shown in figure 5.

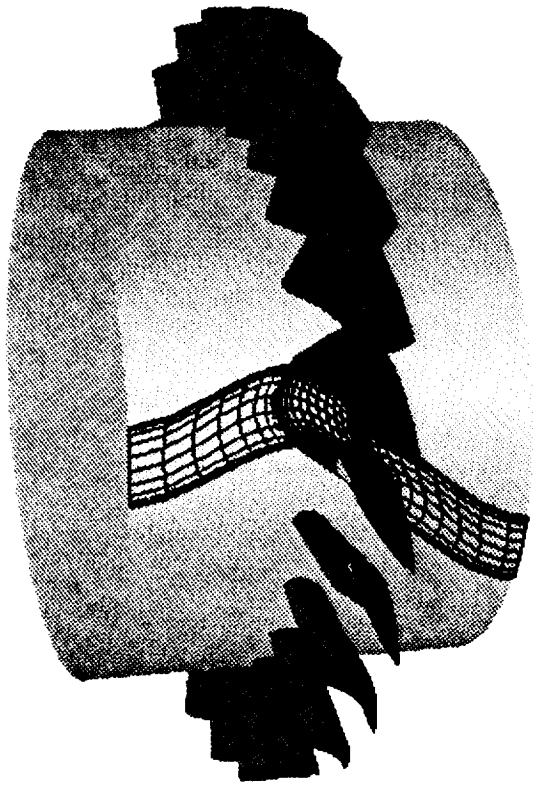


Figure 7.–Representative mesh surfaces for a three-dimensional turbine nozzle (Axial flow device).

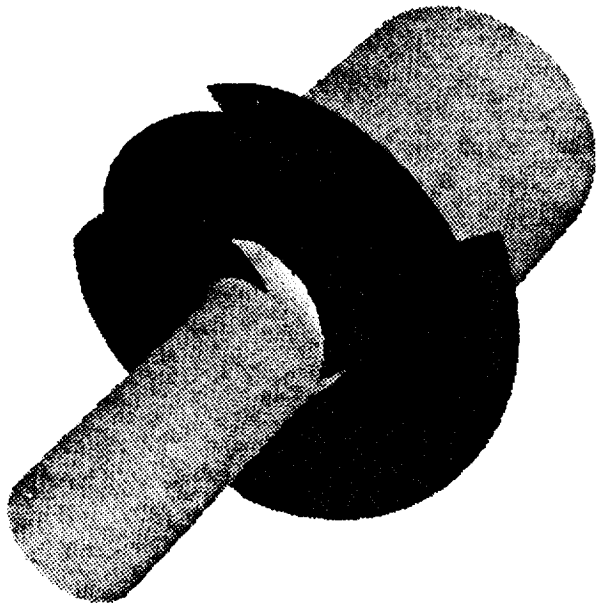


Figure 8.–Representative mesh surface for an inducer (Axial flow device).

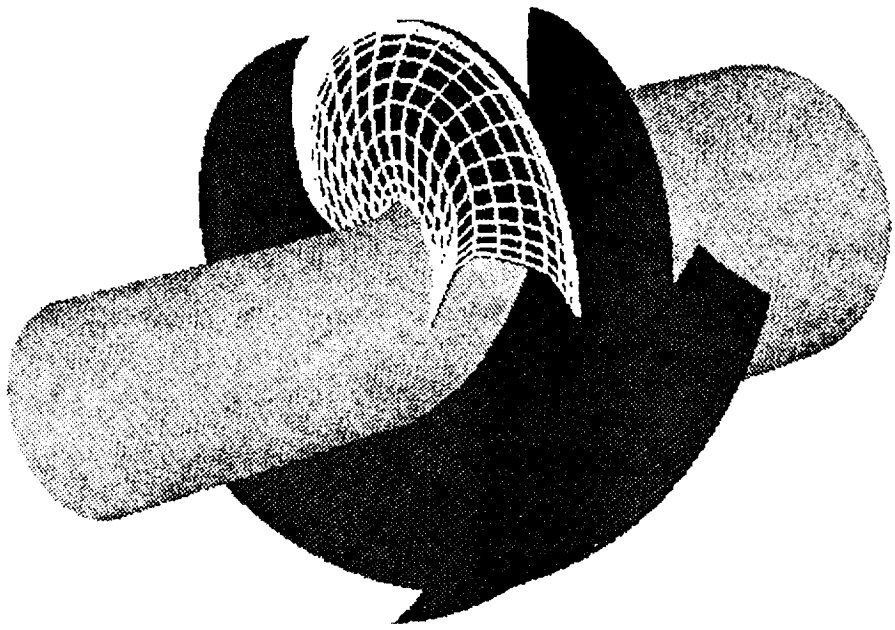


Figure 9.–Representative mesh surface for an inducer with a novel, sickle shaped blade leading edge (Axial flow device).

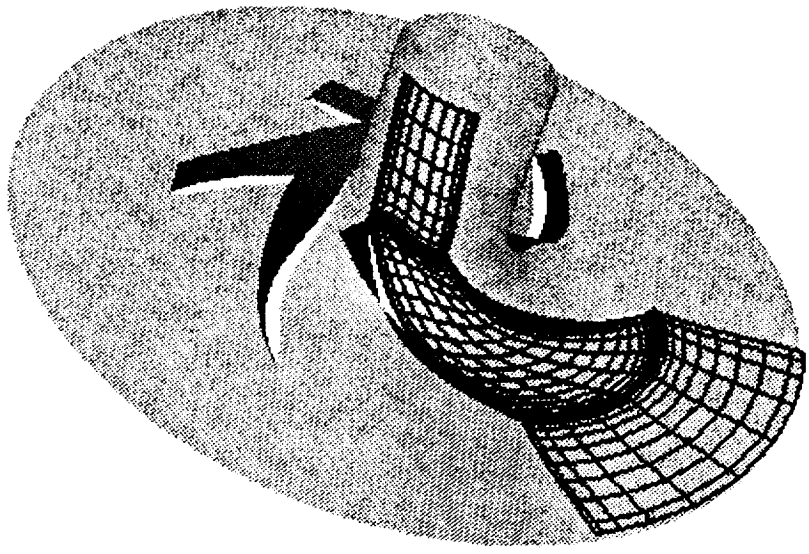


Figure 10.–Representative mesh surfaces for an impeller (Radial flow device).



C-8

N95- 28763

# A GRID GENERATION SYSTEM FOR MULTI-DISCIPLINARY DESIGN OPTIMIZATION

William T. Jones* and Jamshid Samareh-Abolhassani**
Computer Sciences Corporation
Hampton, Virginia

## ABSTRACT

*A general multi-block three-dimensional volume grid generator is presented which is suitable for Multi-Disciplinary Design Optimization. The code is timely, robust, highly automated, and written in ANSI "C" for platform independence. Algebraic techniques are used to generate and/or modify block face and volume grids to reflect geometric changes resulting from design optimization. Volume grids are generated/modified in a batch environment and controlled via an ASCII user input deck. This allows the code to be incorporated directly into the design loop. Generated volume grids are presented for a High Speed Civil Transport (HSCT) Wing/Body geometry as well a complex HSCT configuration including horizontal and vertical tails, engine nacelles and pylons, and canard surfaces.*

## INTRODUCTION

The continuing increase in computer speeds and the advancement of numerical algorithms has helped to increase the interest in Multi-disciplinary Design Optimization (MDO). MDO is a methodology for the design of complex engineering systems and subsystems that coherently exploits the synergism of mutually interacting phenomena. The process involves analyzing independent solutions generated after perturbing the design variables of a baseline system.

Typically, the MDO process proceeds in an iterative manner. Each cycle, at a minimum, includes the generation of numerical solutions, determination of design sensitivities, and system optimization. It is apparent that a large number of cycles may be necessary to complete the optimization, and as a result, the process must be capable of efficient execution in the batch environment with as little human intervention as possible.

Perturbations are based on design sensitivities which are derivatives of design parameters (e.g. lift) with respect to the design variables (e.g. airfoil camber). In practice, design sensitivities from the various disciplines used in the system analysis are obtained separately. These are then combined using the chain rule to obtain the global sensitivity of the system. The derivatives may be obtained analytically, by differentiating the analysis code(s), or numerically, using finite differences. When using finite differences, however, it is often difficult to determine the appropriate step size to be used for a given system.

In many applications of MDO, computational fluid dynamics (CFD) is an integral part of the design process. As a result, it is necessary to possess a rapid and highly automated grid generation capability which produces changes in the surface and volumetric grids to reflect the perturbations of the baseline system. It is also desirable to make grid modifications within the design cycle. Many design parameters, such as lift and drag, are fairly well established after a small number of flow iterations. With geometric changes occurring within the design loop, partially converged flow solutions can be used as a starting solution for the next cycle thereby reducing the overall computational effort[1].

Current grid generation techniques have been strongly shaped by a push to develop interactive tools which aid in the discretization of computational domains. While most of these tools are well suited for the generation of grids about unique configurations, their generality requires a large degree of human

---

* Member of the Technical Staff
** Computer Scientist



PRECEDING PAGE BLANK NOT FILMED

interaction. Even for the smallest changes, like those resulting between MDO cycles, a great deal of input of varying degree is needed to redefine the computational structure. Recently, some tools[2,3] have incorporated useful parametric capabilities, but continue to rely on interactive software control. These methods fall short of satisfying the rapid, "hands-off" grid generation needs of CFD in the MDO cycle.

In this paper, the development of a computer program specifically designed for MDO grid generation is described. The code is capable of generating CFD quality surface and volume grids, as well as analytic grid sensitivities with respect to the design variables. In addition, existing grids generated with other grid generation systems may be modified to reflect the geometric changes defined in an automated design and optimization cycle. This program has been applied in the generation and modification of both inviscid and viscous CFD grids.

In the sections to follow, the approach taken to develop the Coordinate and Sensitivity Calculator for Multi-disciplinary Design Optimization (CSCMDO) computer program is described. Also discussed are results from aerospace test cases involving High Speed Civil Transport (HSCT) configurations and future directions of code development.

## APPROACH

The CSCMDO code includes specialized features required by MDO grid modifications. These features are added to compliment standard volume grid generation methods. The code is controlled via an ASCII user input file for execution in a batch environment. The code developed herein is capable of modifying any of the six faces of a block, representing a computational cube, to reflect geometric changes in the optimized system as defined by input surface(s).

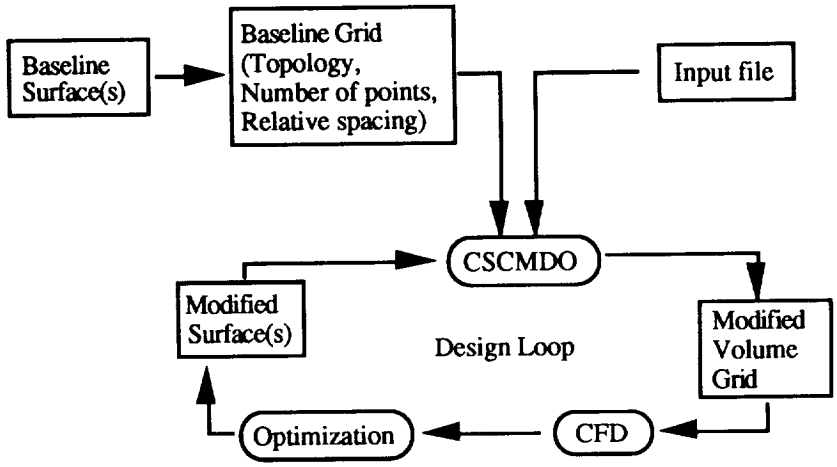



Figure 1 Integration of grid generation into the design loop

Figure 1 shows the layout of a typical MDO design loop. Information input from outside the loop is generated one time before the loop is initiated. This information includes the baseline geometry, baseline CFD grid, and a user input file. The design loop is totally self contained and therefore requires no human intervention. The CSCMDO code operates within the loop to provide automated volume grid generation / modification within the each design cycle.

The baseline surfaces are provided in the form of a structured mesh of discrete point data. The number and distribution of points defining the surfaces are not required to match those of the desired CFD grid. However, sufficient point resolution must be provided so as to adequately define all surface curvature. Some methods of surface modification described herein will require that the surface mesh topology match that of the desired CFD grid. Also, points of interest, such as a wing crank location, must fall along isoparametric lines in the surface definition. Point continuous surface/surface intersections are desirable. In the event that point continuity cannot be provided at surface/surface intersections, intersection curves on each surface should have sufficient resolution such that interpolation errors resulting from the inconsistent data do not contaminate the quality of the resulting CFD grid.

The baseline grid may be in the form of a baseline volume grid or, in some instances, discrete grid point data for the six faces of each block. The original grids can be generated using any structured grid generation package. File formats used are widely used throughout the field of CFD.

Within the design loop, the optimization process is to provide modified surface geometry definition, again as discrete point data[4-6]. The changes represented by the modified surfaces are assumed to be small so as not to violate the original topology definition. In the event the modifications do violate the topology, grid quality checks provide return codes to the software controlling the loop for appropriate action.

Standard algebraic two-dimensional(2D) and three-dimensional(3D) grid generation functionality is implemented for the generation of faces and blocks. Blocks may be automatically subdivided using intermediate face "hard planes" to increase the quality of 3D interpolation. Hard planes may be input or

calculated internally.

## FACE MODIFICATIONS

Face modifications are provided using a variety of methods to be described in the following sections. These methods include simple input of a modified face, parametric updates to a modified surface, projection to a modified surface, and deformations conforming to a modified surface.

### *PARAMETRIC:*

The parametric updates[7] include methods utilizing mappings to uniform parameter space (UPS), as well as arclength parameter space (APS). APS is provided in the event that input surface parameterization is not smooth and orthogonal. The use of UPS with such a surface could produce an unacceptable CFD grid. A detailed explanation is contained in ref. 7. APS seeks to alleviate such shortcomings.

These methods use a forward mapping into the selected parameter space where the grid generation takes place. Parameter values are then mapped back to physical space according to the modified surface.

The mapping to and from both UPS and APS, as well as a description of the parameter space grid generation technique of reference 7, are briefly described here for completeness.

Forward mapping involves the transfer of the surface grid from physical space $\left(\overline{R}=\{X,Y,Z\}^T\right)$ to a parameter space $\left(\overline{W}=\{U,V\}^T\right)$. The surface is a parametric surface such as a bilinear, bicubic, or NURBS representation. UPS is the primary type of parameter space used in grid generation, and is simply constructed using the surface grid indices according to

$$U_{i,j}=i,, \qquad V_{i,j}=j. \quad (1)$$

APS is constructed using the surface arclengths in each computational direction as

$$\begin{aligned} U_{i,j} &= U_{i-1,j}+e_{i,j}, \\ V_{i,j} &= V_{i-1,j}+f_{i,j}, \end{aligned} \quad (2)$$

where

$$\begin{aligned} e_{i,j} &= \left\|\overline{R}_{i,j}-\overline{R}_{i-1,j}\right\|, \\ f_{i,j} &= \left\|\overline{R}_{i,j}-\overline{R}_{i,j-1}\right\|. \end{aligned} \quad (3)$$

Grid generation is the same for both parameter spaces. The edges of the CFD face are determined in the appropriate parameter space. In CSCMDO, the orientation of the CFD face to the background surface is automatically determined. A manual override is provided in the event the orientation search fails. Once the edges are known, the interior of the face is generated using arclength based transfinite interpolation (TFI). With TFI, the interior grid points are defined by the edge and corner contributions according to

$$\begin{aligned} \overline{w}_{l,m} = {} & a_{l,m}\overline{w}_{l,m1}+b_{l,m}\overline{w}_{l,m2}+c_{l,m}\overline{w}_{l1,m}+d_{l,m}\overline{w}_{l2,m} \\ & -a_{l,m}c_{l,m}\overline{w}_{l1,m1}-b_{l,m}c_{l,m}\overline{w}_{l1,m2} \\ & -a_{l,m}d_{l,m}\overline{w}_{l2,m1}-b_{l,m}d_{l,m}\overline{w}_{l2,m2}. \end{aligned} \quad (4)$$

The subscripts $(l1,l2)$ and $(m1,m2)$ are the minimum and maximum indices of the face in the u and v directions respectively. The variables $a,b,c$ and $d$ are the blending functions subject to the following conditions,

$$\begin{aligned} a_{l,m1} = c_{l1,m} = b_{l,m2} = d_{l2,m} = 1, \\ a_{l,m2} = c_{l2,m} = b_{l,m1} = d_{l1,m} = 0. \end{aligned} \quad (5)$$

A very robust set of blending functions has been proposed by Soni[8]. These blending functions are based on the normalized arclength of the edges in physical space, and are defined as

$$\begin{aligned} a_{l,m} &= 1-\eta_{l,m}, \qquad b_{l,m} = \eta_{l,m}, \\ c_{l,m} &= 1-\xi_{l,m}, \qquad d_{l,m} = \xi_{l,m}, \end{aligned} \tag{6}$$

where

$$\begin{aligned} \xi_{l,m} &= \frac{g_{l,m1} + h_{l1,m}\left\{g_{l,m2} - g_{l,m1}\right\}}{P_{l,m}}, \\ \eta_{l,m} &= \frac{h_{l1,m} + g_{l,m1}\left\{h_{l2,m} - h_{l1,m}\right\}}{P_{l,m}}, \\ P_{l,m} &= 1-\left\{g_{l,m2} - g_{l,m1}\right\}\left\{h_{l2,m} - h_{l1,m}\right\}, \end{aligned} \tag{7}$$

The values of $g_{l,m}$ and $h_{l,m}$ are the normalized edge arclengths in physical space. With this interpolation complete the grid can be mapped back from the parameter space to the physical space.

Backward mapping is used to transfer the grid points from the parametric space $\left(\bar{w}_{l,m} = \left\{u_{l,m}, v_{l,m}\right\}^T\right)$ to the physical space $\left(\bar{r}_{l,m} = \left\{x_{l,m}, y_{l,m}, z_{l,m}\right\}^T\right)$. The backward mapping involves three steps. The first step is a search of the surface parameter space to determine the local surface patch which includes the current grid point. The second step is to compute the local patch coordinates $\left(s_{l,m}, t_{l,m}\right)$ for the coordinates in parametric space $\left(\bar{w}_{l,m}\right)$. For UPS this is relatively simple, where

$$\begin{aligned} s_{l,m} &= u_{l,m} - U_{i,j}, \\ t_{l,m} &= v_{l,m} - V_{i,j}. \end{aligned} \tag{8}$$

For APS, the local coordinates are computed using a set of quasilinear equations. The UPS parameter values of $U_{i,j}$ and $V_{i,j}$ are known for each corner of the patch and the patch grid point $\left(u_{l,m}, v_{l,m}\right)$ is known in the interior of the patch. With this information a system of two equations with two unknowns is formed, where

$$\begin{aligned} u_{l,m} &= \left(1-s_{l,m}\right)\left(1-t_{l,m}\right)U_{i,j} + s_{l,m}\left(1-t_{l,m}\right)U_{i+1,j} \\ &\quad +\left(1-s_{l,m}\right)t_{l,m}U_{i,j+1} + s_{l,m}t_{l,m}U_{i+1,j+1}, \\ v_{l,m} &= \left(1-s_{l,m}\right)\left(1-t_{l,m}\right)V_{i,j} + s_{l,m}\left(1-t_{l,m}\right)V_{i+1,j} \\ &\quad +\left(1-s_{l,m}\right)t_{l,m}V_{i,j+1} + s_{l,m}t_{l,m}V_{i+1,j+1}. \end{aligned} \tag{9}$$

The solution to these equations is obtained using the Newton-Raphson method and gives the local coordinates of the patch $\left(s_{l,m}, t_{l,m}\right)$. The final step is to compute the physical coordinates $\left(\bar{r}_{l,m}\right)$ of the grid point on the local surface patch using bilinear or bicubic interpolation.

*PROJECTION:*

Projection of a face allows for the generation of a CFD grid over several surfaces, while also avoiding problems associated with surface parameterization mentioned in the previous section. The projection algorithm[9] used in CSCMDO is described briefly for completeness. Again the surface is approximated by a parametric surface as,

$$\begin{aligned} \bar{R}(\bar{w}) &= \{x(\bar{w}), y(\bar{w}), z(\bar{w})\}^T, \\ \bar{w} &= \{u, v\}^T. \end{aligned} \tag{10}$$

The process of projecting a point, $\bar{r}$, onto a surface $\bar{R}(\bar{w})$, involves finding $\bar{w}$ such that the distance, $d$, between $\bar{r}$ and $\bar{R}(\bar{w})$ is minimized such that it does not violate the limits of $\bar{w}$. The distance, $d$, can be written in terms of parameter $\bar{w}$ as,

$$d^2(\bar{w}) = f(\bar{w}) = \left|\bar{R}(\bar{w}) - \bar{r}\right|^2. \tag{11}$$

To minimize $d$, Eq. (11) must be minimized with respect to $\bar{w}$. This is accomplished by setting the gradient of $f$, $\nabla f(\bar{w})$, equal to zero, as

$$\begin{aligned} \nabla f(\bar{w}) = G_i(\bar{w}) &= \frac{\partial f(\bar{w})}{\partial u_i} = 0, \\ G_i &= \frac{\partial \bar{R}(\bar{w})}{\partial u_i} \cdot \left\{\bar{R}(\bar{w}) - \bar{r}\right\}. \end{aligned} \tag{12}$$

Solution of the above nonlinear system of equations for $\bar{w}$ results in the projected point.

As with the parametric update, a surface is represented as a set of bilinear or bicubic patches. For each patch, $\bar{R}(u,v)$, is approximated in terms of its parameters as,

$$\begin{aligned} \bar{R}(u,v) = (1-u)(1-v)\bar{R}_{i,j} + u(1-v)\bar{R}_{i+1,j} \\ +(1-u)v\bar{R}_{i,j+1} + uv\bar{R}_{i+1,j+1}. \end{aligned} \tag{13}$$

Combining Eqs. (12) and (13) yields the following system of nonlinear equations,

$$\frac{\partial \bar{R}}{\partial u} \cdot \left\{\bar{R}(\bar{w}) - \bar{r}\right\} = 0, \quad \frac{\partial \bar{R}}{\partial v} \cdot \left\{\bar{R}(\bar{w}) - \bar{r}\right\} = 0, \tag{14}$$

where,

$$\begin{aligned} \frac{\partial \bar{R}}{\partial u} &= (1-v)\left(\bar{R}_{i+1,j} - \bar{R}_{i,j}\right) + v\left(\bar{R}_{i+1,j+1} - \bar{R}_{i,j+1}\right), \\ \frac{\partial \bar{R}}{\partial v} &= (1-u)\left(\bar{R}_{i,j+1} - \bar{R}_{i,j}\right) + u\left(\bar{R}_{i+1,j+1} - \bar{R}_{i+1,j}\right). \end{aligned} \tag{15}$$

Again the system is solved by the Newton-Raphson method. An additional method of projection available in CSCMDO is the projection onto an original surface(s) and evaluation based on a new surface(s). This method obtains the parametric coordinates by projecting each face grid point onto the baseline surface. The parametric coordinates are then mapped to the physical space using the surface(s) driving the modification. The resulting grid maintains the same characteristics of the original grid. The shortcoming of this method, however, is the required existence of both the original and modified surfaces at the time of projection.

*DEFORMATION:*

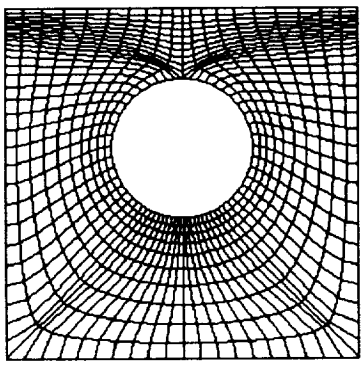
Deformation of a face is key in providing reusability of an existing CFD grid. When a face is modified, a comparison to the original face edge points is made according to,

$$\Delta\bar{R} = \bar{R}_{modified} - \bar{R}_{original}, \tag{16}$$

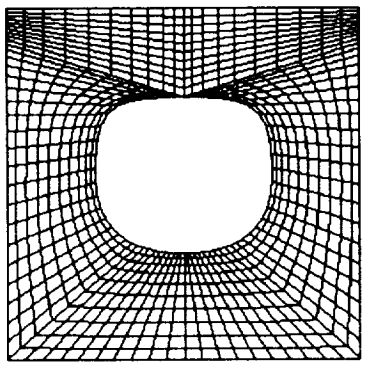
Rearranging Eq. (16), the modified edge is given by,

$$\overline{R}_{modified} = \overline{R}_{original} + \Delta\overline{R}. \qquad (17)$$

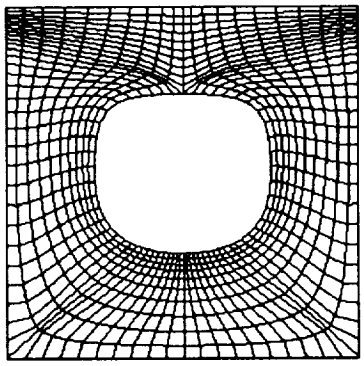
If all edges have been modified, Eqs. (16) and (17) are applied to the interior grid points to yield the modified face. In the event that less than 4 edges of a face are modified, unmodified edge $\Delta\overline{R}$ values in Eq. (17) are obtained by blending the ending values of $\Delta\overline{R}$ along the edge. The edge $\Delta\overline{R}$ values are then interpolated into the interior for use in Eq. (17). This method maintains the grid quality characteristics of the original CFD grid and provides time saving reusability of an optimal CFD grid.



a) Original smoothed, orthogonal face



b) Modified face with algebraic TFI



c) Modified face with face deformations

Figure 2 Example of Face deformation

Figure 2 shows the method applied to a C-H grid around a circle enclosed in a rectangular outer domain. The original grid in figure 2-a shows smoothness and orthogonality characteristics obtained through partial differential equation (PDE) smoothing. Figures 2-b and 2-c show a modification to the circular inner edge. In Figure 2-b the edge is modified and the face is re-initialized with arclength based TFI. Figure 2-c shows the effect of a face deformation governed by the same change to the circular edge. Note the preservation of the smoothness and orthogonality characteristics in figure 2-c. This method is extended to 3D as will be discussed in later sections.

## BLOCK MODIFICATIONS

Volume modifications are accomplished using standard algebraic methods for the re-initialization of the block interior, or by deformation of the original block interior based on changes defined by the six faces.

### *GRID GENERATION:*

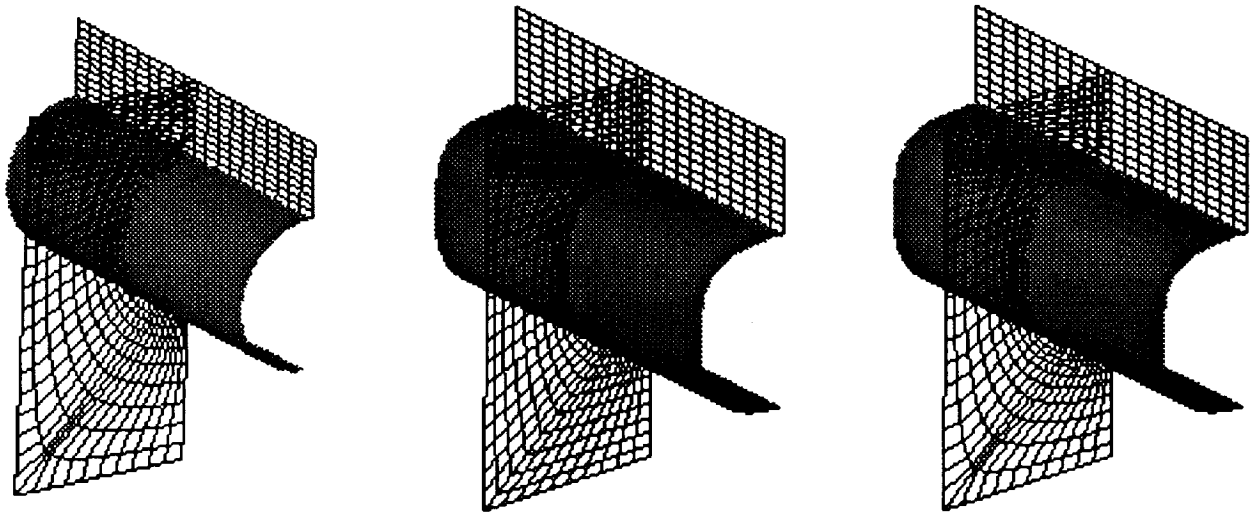
The generation of the block interior is conducted in a manner which is an extension of the 2D methods of arclength based TFI described above. As previously mentioned, a block may be automatically subdivided by inserting intermediate faces, or "hard planes" within the block. Each hard plane acts to subdivide the block providing added control over the quality of developing interior. Hard planes may be directly input, as in the case where a PDE smoother was used to obtain an optimal grid, or generated internally using the 2D methods described above.

### *DEFORMATION:*

A method similar to that described for face deformation is extended to 3D for use in the deformation of entire blocks. Once the deformations are calculated for each of the six faces of a block, arclength based TFI is used to blend the deformations

into the block interior. Deformations are then applied to the original grid points to obtain a deformed block. Again this method of deformation facilitates the reusability of an existing volume grid. The overall quality and characteristics of an existing block are maintained after face modification in the modified block.

Figure 3 shows the 3D deformation applied to the same C-H topology used previously in 2D. Figure 3-a shows smoothness and orthogonality characteristics obtained through partial differential equation (PDE) smoothing for the original block. Figures 3-b and 3-c show the same modification to the cylindrical inner face. In Figure 3-b the face is modified with the adjacent faces and interior re-initialized with arclength based TFI. Figure 3-c shows the effect of face and block deformation governed by change to the cylindrical inner face. Again note the preservation of the smoothness and orthogonality characteristics in figure 3-c on the block interior.



a) Original smooth orthogonal block b) Modified with algebraic TFI c) Modified with deformation

Figure 3 Example of block deformation

## GRID QUALITY

A check of grid quality measures is included in the code for diagnostic output as well as limited control when running in a batch environment. Cell volume and cell skewness are calculated on a block by block basis. A histogram of cell skewness is provided with the code output showing the percentage of the grid possessing a skewness over a range of $0^{o}$ to $90^{o}$. Minimum cell volume, maximum cell skewness, and maximum average cell skewness controls are provided to trigger return codes which are used by the process controlling batch execution. The return codes allow for the controlling process to exit the design loop without wasting continued resources in the event the resulting volume grid is not suitable for use.

## SENSITIVITY

Calculation of grid sensitivity to the design variables involves the automatic differentiation of the CSCMDO code. This is accomplished using the Automatic DIfferentiation of C source (ADIC) processor from Argonne National Labs. ADIC is a tool for the automatic differentiation of ANSI C programs providing similar capabilities for C as ADIFOR[10] does for FORTRAN77. The ADIC tool operates on a given source code using a specified set of dependent and independent variables, to produce an augmented C source that computes not only the original scalar result, but also the partial derivatives of all of the specified dependent variables with respect to the independent variables. ADIC employs the Sage++ programming environment developed at Indiana University.

The cost of generating the sensitivities with the ADIC processed version of CSCMDO (CSCMDO.AD) has

proven to be negligible when compared to the overall cost of a CFD design cycle. With an ADIC processed source code, the ratio between the cost of evaluating a gradient with n components and the cost of evaluating the original underlying scalar function is not n+1, but is bounded by 5 at most[11]. Preliminary testing of CSCMDO.AD has produced reasonable results. These results have been compared to finite difference calculations and show favorable agreement.

VISCOUS GRID CAPABILITIES

The methods presented herein have been successfully applied to viscous grids. Care must be taken to provide surface geometry containing sufficient resolution to capture surface curvature. The use of block deformation is very useful in the reuse of viscous grids which were originally refined using PDE techniques.

## RESULTS

CSCMDO has been extensively tested on aerospace configurations at the NASA Langley Research Center's (LaRC) Geometry Laboratory (GEOLAB). The test cases range from simple wing/body configurations to full HSCT geometry with tail surfaces, engine nacelles, and canards. The code is also used outside of the design loop in the GEOLAB for the rapid modification and quality check of existing CFD volume grids.

Execution time is difficult to determine for the code due to the wide variety of execution schedules available to the user. Typical timing, however, can be given as an example of two types of configurations.

The first is that of a simple double delta wing/body aircraft. The computational domain was broken into two C-H block with a total of 350,000 grid points. Seven modified geometries were supplied along with the baseline and the block topology. The geometry modifications included changes to wing sweep, camber, span, crank location, etc.. All eight volume grids, baseline and seven modifications, were generated sequentially on a Silicon Graphics Onyx workstation in double precision in under three minutes.

Figure 4 Complex HSCT configuration

The second case is that shown in figure 4. The configuration includes a double delta wing, tail surfaces, two flow through engine nacelles with a plug insert, engine pylons, and a canard. The computational domain contained seventeen blocks with at total of over 1.5 million grid points. This complex case required on the order of three minutes for the generation of a single complete volume grid with modifications to a variety of surfaces.

Figure 5 presents the results of the case involving the complex HSCT in figure 4. Figure 5-a shows every third grid line in each computational direction on the surface of the configuration. A cut from the 3D volume is shown in figure 5-b. The view is looking downstream at the nose of the aircraft.

## FUTURE DIRECTIONS

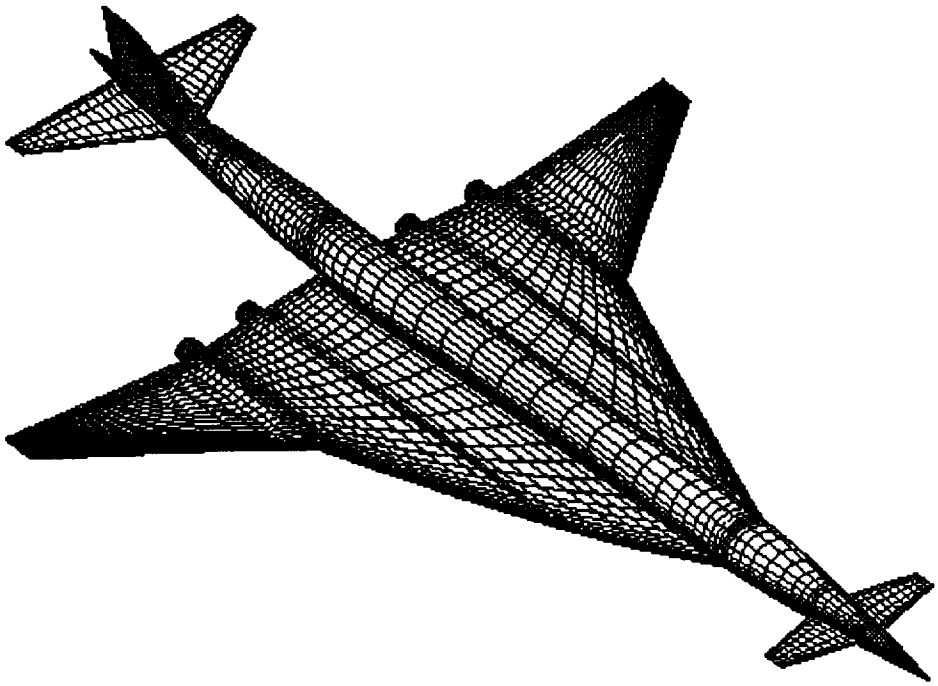
Future plans include the addition of PDE smoothing of block faces and entire blocks within the CSCMDO code. Also to be included are capabilities for complete domain decomposition within the code. Presently the block topology, number of points, and relative spacing are required input to CSCMDO. Future plans are to incorporate methods of defining these key features of a computational domain within the program.

## CONCLUSIONS

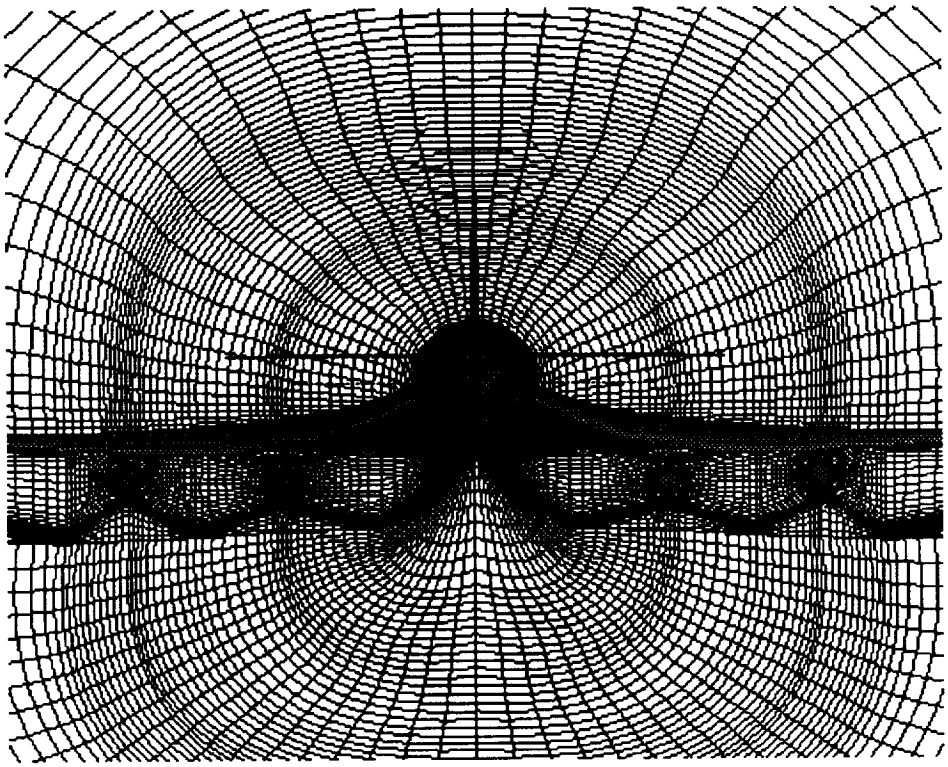
A CFD volume grid generator has been presented for which is designed around the needs of multi-disciplinary design optimization. The code employs methods which make it ideal for the type of grid modifications required by MDO. The code is robust, timely, platform independent, and ready to meet the needs of multi-disciplinary design.

## ACKNOWLEDGMENTS

The authors would like to acknowledge the assistance provided by Christian Bischof and Andrew Mauer of Argonne National Laboratories. Their help in the area of automatic differentiation, and in the processing of CSCMDO with the ADIC tool, was crucial to the timely implementation of grid sensitivity calculations in CSCMDO.



a) CFD surface grid for complex geometry
(Note: every 3rd grid line drawn)



b) Grid from cut normal to flow

Figure 5 Examples of complex volume grid

## References


1) Doria, Michael L., "An Investigation of Design Optimization using a 2-D Viscous Flow Code with Multigrid", NASA/ASEE Summer Faculty Fellowship Program, 1990.

2) Steinbrenner, John P., and Chawner, John R., "The GRIDGEN Version 9 Multiple Block Grid Generation Software", MDA Engineering Report 94-01, 1994.

3) Bertin, D., Casties, C., and Lordon, J., "A New Automatic Grid Generation Environment for CFD Applications", Fourth International Conference on Numerical Grid Generation in CFD and Related Fields, Swansea, Wales, 1994, pp. 391,402.

4) Barger, Raymond L., and Adams, Mary S., "Automatic Computation of Wing-Fuselage Intersection Lines and Fillet Inserts with Fixed-Area Constraint", NASA TM-4406, March 1993.

5) Barger, Raymond L., and Adams, Mary S., "Automatic Computation of Euler-Marching and Subsonic Grids for Wing-Fuselage Configurations", NASA TM-4573, July 1994.

6) Barger, Raymond L., and Adams, Mary S., "Automatic Procedures for Computing Complete Configuration Geometry from Individual Component Descriptions", NASA TM-4607, July 1994.

7) Samareh-Abolhassani, Jamshid, and Stewart, John E., "Surface Grid Generation in a Parameter Space", Journal of Computational Physics, Vol. 113, No. 1, July 1994.

8) Soni, B.K., "Two- and Three-Dimensional Grid Generation for Internal Flow Applications of Computational Fluid Dynamics", AIAA-85-1526, 1985.

9) Samareh-Abolhassani, Jamshid, "Unstructured Grid on NURBS SURFACES", AIAA-93-3454, 1993.

10) Bischof, Christian, Carle, Alan, Corliss, George, Griewank, Andreas, and Hovland, Paul, "ADIFOR: Generating Derivative Codes from FORTRAN Programs", Scientific Programming, Volume 1, Number 1, pp. 11,29, 1992.

11) Griewank, Andreas, "On Automatic Differentiation", Mathematics and Computer Science Division Preprint ANL/MCS-P10-1088, November, 1988.

# NEW METHODS/APPROACHES/ APPLICATIONS (2)

PRECEDING PAGE BLANK NOT FILMED

N95- 28764

# AUTOMATIC MULTI-BLOCK GRID GENERATION FOR HIGH-LIFT CONFIGURATION WINGS*

Byoungsoo Kim and Scott Eberhardt
Department of Aeronautics and Astronautics
University of Washington
Seattle, WA 98195

## SUMMARY

A new method for automatic multi-block grid generation is described. The method combines the Modified Advancing Front Method as a Predictor with an elliptic scheme as a corrector. It advances a collection of cells by one cell height in the outward direction using Modified Advancing Front Method, and then corrects newly-obtained cell positions by solving elliptic equations. This predictor-corrector type scheme is repeatedly applied until the field of interest is filled with hexahedral grid cells. Given the configuration surface grid, the scheme produces block layouts as well as grid cells with overall smoothness as its output. The present method saves human-time and reduces the burden on the user in generating grids for general 3-D configurations. It was used to generate multi-block grids for wings in their high-lift configuration.

## INTRODUCTION

Grid generation is an essential and critical task for successful application of CFD in engineering process. Grid generation has been, and still is, a labor-intensive and expertise-requiring task, especially for complicated configurations. Configurations for which grid generation requires enormous amount of human-time and labor can be easily found: Automobiles, underwater vehicles, and airplanes. One example of further complication is that modern flying vehicles use high-lift devices, such as slats and multiple flaps, during take-off and landing. Flow fields around such complicated configurations are generally decomposed into several relatively-simple sub-domains. Once sub-domains are identified, it is relatively simple and straightforward to fill each of them with structured meshes.

A sub-domain may either share a common interface with immediately adjacent sub-domains or be overlaid with each other. The first strategy of using sub-domains with common interfaces is the subject of this research, and is called a multi-block grid approach. Decomposition of physical flow fields into blocks, so-called block generation, is the most challenging task in the process of multi-block grid generation. Development of an automatic block generation algorithm is one of several highly-pursued topics in the field of grid generation. (ref. 1 to 11). Many of existing block generation schemes generate block boundaries in their topological form. In other words, once the user is given those block boundaries, he/she should distribute grid points along those block boundaries and generate internal grid points for each block to obtain final multi-block grids.

In this paper, a new approach for automatic multi-block grid generation is presented. The new approach generates multi-block grids in a single step, instead of generating block boundaries first and then filling each block with a certain number of grid points specified by the user later. For realization of this unique feature, the user provides configuration surface grids and the grid spacing in the advancing direction as input to the scheme. These user-specified surface grids are undoubtedly composed of several blocks, especially for complicated configurations. This speculation is based on not only the fact that it is

* Work partially funded by Joint Research Interchange from NASA Ames Research Center

PRECEDING PAGE BLANK NOT FILMED

very difficult and undesirable to discretize a complicated configuration surface with a single structured grid but also the fact that the surface is likely to be described as a collection of relatively-simple segments by geometry definition programs or CAD programs and the user may have, according to the expected flow phenomena, a preference on the grid point distribution along the surface which is more easily accomplished by the use of multi-block surface grid. These user-specified surface grids will be the actual grid points along the block boundaries in the resulting multi-block volume grids, and play an active role in generating final grids as explained in the following section.

## METHOD

The new method uses the combination of the Modified Advancing Front Method as a Predictor with an elliptic scheme as a corrector (MAP scheme). Kim introduced the MAP scheme for 2-D and used it for multi-block grid generation around 2-D configurations (ref. 12). The MAP scheme advances a collection of surface grid blocks by one cell height at a time. In this method, the collection of surface blocks which advances simultaneously is called a "front".

### MAP scheme

As the name implies, the MAP scheme (an acronym of the Modified Advancing Front Method as a Predictor with an elliptic scheme as a corrector) is a predictor-corrector approach. A front is advanced by the predictor and smoothed by the corrector. Each front goes through this predictor-corrector step until the flow field of interest is filled with multi-block grids.

Predictor step.-It advances a front using the Modified Advancing Front Method. The Advancing Front Method (AFM) was originally developed for unstructured triangular meshes (ref. 13 to 15). In the present research, however, this method has been modified for hexahedral mesh generation. The Modified AFM not only enables simultaneous generation of a collection of hexaheral cells but also adjusts the distance and direction of advancement of each grid point according to the surrounding situation. It first interpolates the distance and direction of advancement for each internal point of a front from those values along the boundary of the front. Then, all the points on the front are advanced, resulting in a new front, and the same number of hexahedral cells as that of quadrilateral cells along the old front are obtained. In providing a surface grid as initial fronts, (i,j) indexing of each surface block should be ordered in a consistent manner so that its outward direction can be readily identified; e.g. right-handed rule to have the third k-axis as the outward direction, $\hat{e}_i \times \hat{e}_j \propto \hat{e}_k$. The new front, however, usually carries the non-smoothness, if any, of the old front, and magnifies it, making further advancement impossible or meaningless, especially for concave regions. This situation is avoided by adopting a corrector step.

Corrector step.-In the corrector step, Laplace's equations are used as the elliptic equations by following Cordova's approach (ref. 16). To apply the elliptic corrector, an image of the old front with respect to the new front is first introduced. Using these 3 fronts, 3-D Laplace equations are solved along the new front with the other two as fixed boundary conditions. This procedure is illustrated in Fig. 1. In this example, the initial front is composed of 3 blocks of surface grid, as shown in Fig. 1(a). The predicted front by the Modified AFM is shown in Fig. 1(b). The image front and the resultant 3-front system are shown in (c) and (d) of Fig. 1, respectively. As a solution to the elliptic equations, a smoother distribution of grid points for the new front is obtained, as shown in Fig.1(f), compared to that of the predicted front, as reproduced in Fig. 1(e).

Advancement of fronts.-The distance and direction of advancement of each front is pre-decided by the user ("free advancement"), except for the fronts which have adjacent fronts aligned in the direction of their own advancement ("guided advancement"). If a front falls into the second category, the distance and direction of advancement are decided by the adjacent fronts. The user-specified surface blocks can be grouped into as many number of fronts as the user wants, and the way they are grouped affects the structure of final multi-block grids. Examples of different structures of final volume grids through different setting of initial fronts are demonstrated for a simple hexahedral body in Fig. 2, Fig. 3, and Fig. 4. In Fig. 2,

the 6 block surface grid of the hexahedral body is taken as a single front. As a result, the MAP scheme produces a 6 block grid, which shows a very smooth distribution of grid points along the outer block boundaries. On the other hand, Fig. 3 shows a different block structure due to different arrangement of initial fronts. In this example, the body surface blocks are grouped into 2 fronts; one front with 5 blocks and the other with one block. The two fronts are shown separated in Fig. 3(a). The resultant volume grid has 10 blocks (Fig. 3(b)), and the outer block boundaries are shown in (c). Fig. 4 shows a 26 block grid for the same configuration. In this example, each of 6 surface blocks is taken as a front of its own. In the results presented in this paper, two adjacent blocks are connected in the "complete" sense. In other words, they share an exactly same interface as one of their block faces. Adoption of the "complete" inter-block connectivity makes the total number of blocks in the resulting volume grids higher than that with the "partial" connectivity.

### Bridging of distant fronts

For multi-body configurations such as high-lift wings, two distant fronts can march toward each other. One example is shown in Fig. 5: One front from the main wing and the other from the flap advance toward each other. If this situation is not dealt with properly, the two fronts will pass each other and generate hexahedral cells in the region which is already occupied by other cells. If two fronts are found advancing toward each other and less than a specified distance away, then the scheme will "bridge" the two with a third front. To bridge two distant fronts, proper grid lines along each front, "bridging edges", are first identified (A-B and C-D in Fig. 5).

Once the bridging edges are obtained, they are used to form a 3-D surface grid surrounded by the bridging edges and additional edges connecting them (A-C and B-D). This newly-obtained surface grid is called a "bridging" front. In general, the pair of bridging edges have a different number of grid points with respect to each other. Fig. 6 shows bridging fronts of different structures due to the difference in the number of grid points along the bridging edges and along the edges connecting them. With the help of the "bridging" front, the two distant fronts which were to collide with each other are connected with each other, and the advancement of those fronts will be adjusted accordingly. The sequential advancement of fronts continues until the user-specified outer boundary is reached. The scheme groups hexahedral cells properly, resulting in a multi-block volume grid system with "complete" inter-block connectivity.

## EXAMPLES

The first example is for a generic 3-D single-element wing. NACA 0012 airfoil is used as its wing section, except the round wing tips which are formed by rotating the wing section at the last spanwise station, resulting in a half body of revolution. Fig. 7(a) shows the user-specified wing surface grid together with a surface grid for the wake-plane. Due to its geometrical shape, it is natural and easy for the user to generate the surface grid for the wing-tip region in such a way that all the grid lines in the chordwise direction converge into a single point on each end of the half body of revolution. The user doesn't have to provide the wake-plane grid. By providing it as a part of input, however, he/she can be assured to have the grid point distribution along the wake the way he/she wants. As explained earlier, a "front" is supposed to advance in only one direction. This means that the wake-plane grid is composed of two sets of exactly same grids glued to each other back to back. One set has a reversed ordering of index (i,j) with respect to that of the other. The initial surface grid is composed of 6 surface blocks with "complete" connectivity; one on the upper side of the wing, one on the lower side of the wing, two on both wing tips, one on the upper side of the wake-plane, and the last on the lower side of the wake-plane. By taking those 6 block surface grid as a single "front" and specifying the grid spacing in the advancing direction, the MAP scheme generates 6 block volume grid. The block edges of the resultant grid system are shown in Fig. 7(b). The block boundary grid lines along the symmetry plane are also included. Generating this grid system can be thought of as the analogy of blowing a balloon with the initial shape of Fig. 7(a). The outer block boundaries, shown in Fig. 7(c), and the internal block boundaries, shown in Fig. 7(d), indicate that the grid point distribution becomes smoother and more evenly-spaced as the front advances outward.

In the previous example, the block structure of the resulting volume grid is naturally decided by the block structure of the initial surface grid. With the MAP scheme, however, totally different block structures of volume grids can be easily obtained from the same initial surface grid by changing the arrangement of initial fronts, and it is demonstrated in the next example. Fig. 8(a) shows the exactly same surface grid as Fig. 7(a). In this example, the 6 block surface grid is divided into 3 fronts. One front has 4 surface blocks; the upper wing surface, the lower wing surface, the upper wake surface, and the lower wake surface. The other two fronts have one surface block from each wing tip. In this example, the two wake-plane blocks do not have connectivity at both spanwise ends. The MAP scheme generates a 14 block volume grid as an output. Fig. 8(b) shows the block edges. The outer block boundaries and the internal ones are shown in Fig. 8(c) and (d). These first two examples show the flexibility of the MAP scheme.

The next example is for a simplified high-lift configuration wing. Fig. 9(a) shows a 6 block configuration surface grid; 3 blocks on the main wing and 3 on the flap. The wing tip of the flap is flat while that of the main wing is round. The gap between the main wing and the flap is filled with a "bridging" front, as shown in Fig. 9(b). The resultant 28 block volume grid is shown in Fig. 9(c) and (d). In this example, the number of grid points along the leading edge of the flap is same as the number of points along the trailing edge of the wing. This situation is not always true. The present wing-flap configuration is a simplified and specific one, i.e. a full-length flap. One step toward generalization with the same configuration is to use a different number of grid points in the spanwise direction for the wing and the flap.

Fig. 10(a) shows the same configuration wing as Fig. 9(a). In this example, however, the flap has more grid points in the spanwise direction than the previous case. The "bridging" front should be formed in accordance to the number of grid points along the bridging edges, and the result is shown in Fig. 9(b). Once distant fronts are bridged with the help of a bridging front, the MAP scheme advances each front until the outer boundary is reached. Block edges and outer block boundaries of the resulting 38 block volume grid are shown in Fig. 9(c) and (d) respectively.

## CONCLUSION

The MAP scheme can generate multi-block grids in an automatic fashion. As an input for the scheme the user provides a body surface grid and the grid spacing in the advancing direction. The user can choose how he/she wants to group the surface blocks to form initial fronts. The present scheme produces block layouts as well as grid cells of smooth distribution as its output. Due to its unique approach, the present method saves human-time and reduces the burden on the user in generating grids for general 3-D configurations. The configuration of the high-lift wings shown in this paper, however, is not general enough. Generalization and improvement of the grid generation program using the MAP scheme for realistic 3-D configurations is currently underway.

## REFERENCES


1. Schonfeld, T., and Weinerfelt, P., "The Automatic Generation of Quadrilateral Multi-Block Grids by the Advancing Front Technique," Proceedings of the 3rd International Conference on Numerical Grid Generation in CFD, Barcelona, Spain, June 3-7, 1991.

2. Stewart, M.E.M., "Domain-Decomposition Algorithm Applied to Multielement Airfoil Grids," AIAA Journal, Vol. 30 No. 6, pp1457-1461, June, 1992.

3. Andrews, A.E., "Knowledge-Based Flow Field Zoning," *Numerical Grid Generation in Computational Fluid Mechanics '88*, Pineridge Press, pp 13-22, 1988.

4. Weatherill, N.P., and Shaw, J.A., "Component adaptive grid generation for aircraft configurations," AGARD-AG-309, pp 29-39, 1988

5. Shaw, J.A., Georgala, J.M., and Weatherrill, N.P., "The Construction of Component-Adaptive Grids for Aerodynamic Geometries," *Numerical Grid Generation in Computational Fluid Dynamics '88*, Pineridge Press, pp383-394, 1988

6. Georgala, J.M., and Shaw, J.A., "A Discussion on Issues Relating to Multiblock Grid Generation," AGARD-CP-464, 1990.

7. Allwright, S.E., "Techniques in Multiblock Domain Decomposition and Surface Grid Generation," *Numerical Grid Generation in Computational Fluid Dynamics '88*, Pineridge Press, pp 559-568, 1988

8. Allwright, S., "Multiblock Topology Specification and Grid Generation for Complete Aircraft Configurations," AGARD-CP-464, 1990.

9. Dannenhoffer, J.F. III, "A Block-Structuring Technique for General Geometries," AIAA 91-0145, Reno, January 1991.

10. Dannenhoffer, J.F. III, "A New Method for Creating Grid Abstractions for Complex Configurations," AIAA 93-0428, Reno, January 1993

11. Noble, S.S., and Cordova, J.Q., "Blocking Algorithms for Structured Mesh Generation," AIAA92-0659, 1992

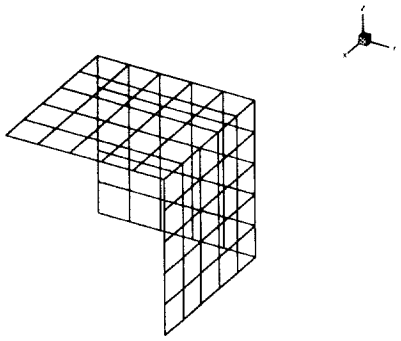
12. Kim, B, Automatic Multi-Block Grid Generation about Complex Geometries, PhD. Dissertation, University of Washington, 1994

13. Lo, S.H., "A New Mesh Generation Scheme for Arbitrary Planar Domains," Int. J. Num. Meth. Eng., Vol. 21, pp.1403-1426, 1985.

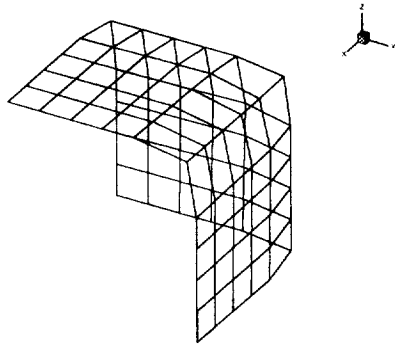
14. Peraire, J. et al, "Adaptive Remeshing for Compressible Flow Computations," J. Comp. Phys. 72, PP.449-466, 1987.

15. Lohner, R., and Parikh, P., "Generation of Three-Dimensional Unstructured Grids by the Advancing Front Method," AIAA Paper 88-0515, Reno, January 1988.

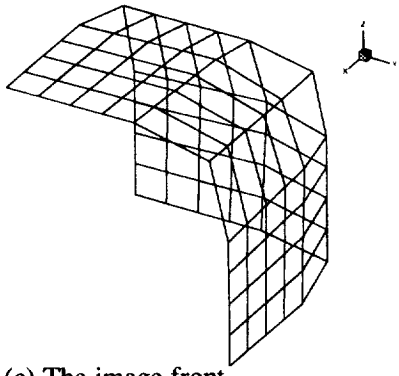
16. Cordova, J.Q.,"Advances in Hyperbolic Grid Generation," 4th International Symposium on Computational Fluid Dynamics, Davis, CA, Sep. 1991



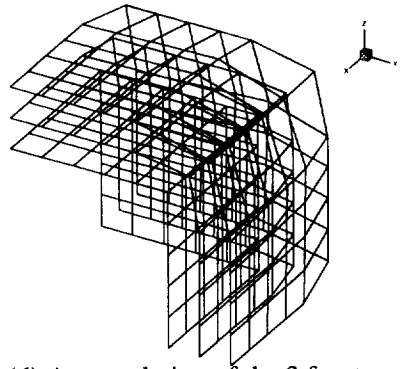
(a) The old front



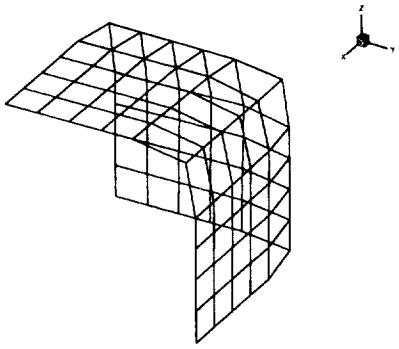
(b) The predicted new front



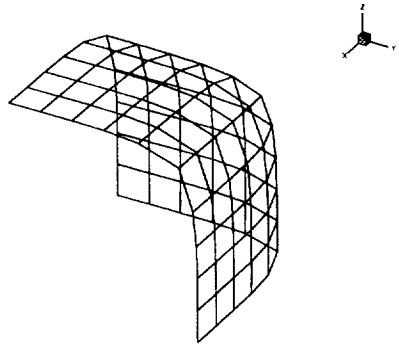
(c) The image front



(d) Accumulation of the 3 fronts

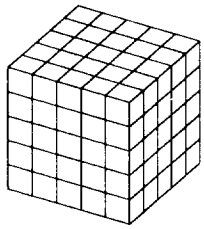


(e) The predicted new front, (b)

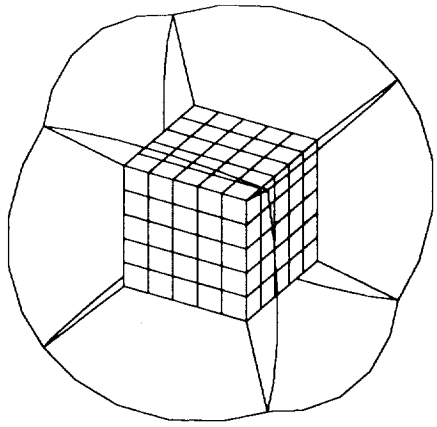


(f) The corrected new front

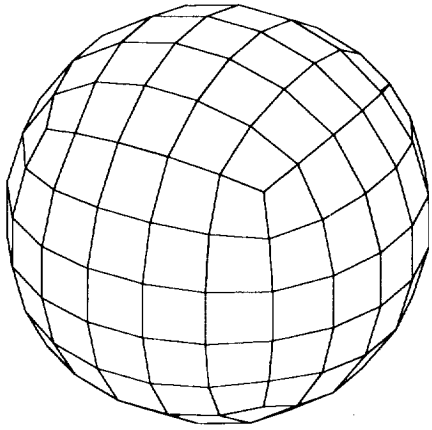
Fig. 1 Procedure of the predictor-corrector approach



(a) Initial front with 6 block surface grid

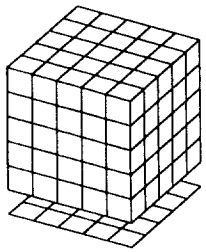


(b) Block edges of the resulting 6 block volume grid

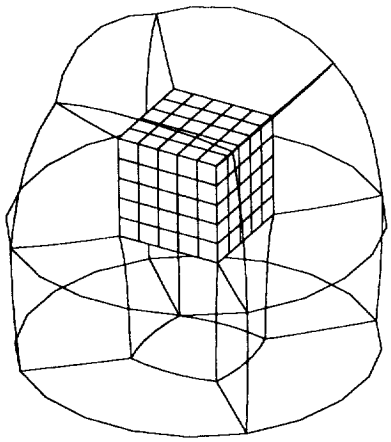


(c) Outer block boundaries

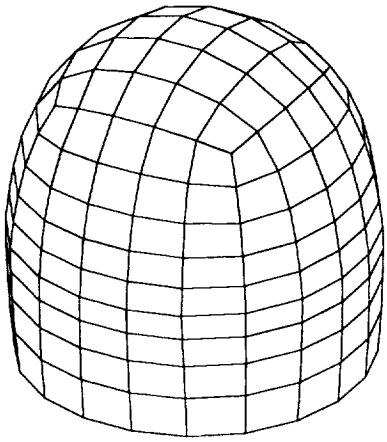
Fig. 2 6 block volume grid generation about a hexahedral body



(a) 2 initial fronts: One with 5 surface blocks and the other with 1 block

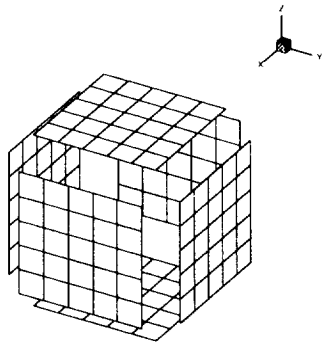


(b) Block edges of the resulting 10 block volume grid

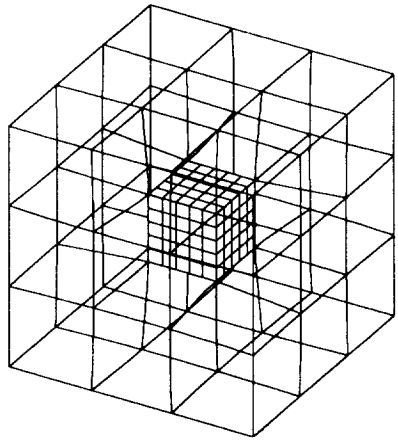


(c) Outer block boundaries

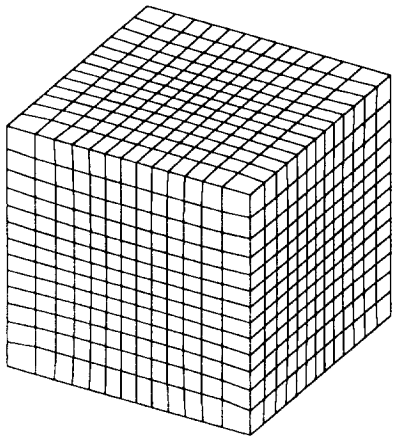
Fig. 3 10 block volume grid generation about a hexahedral body



(a) 6 initial fronts: Each with 1 surface block from each face



(b) Block edges of the resulting 26 block volume grid



(c) Outer block boundaries

Fig. 4 26 block volume grid generation about a hexahedral body

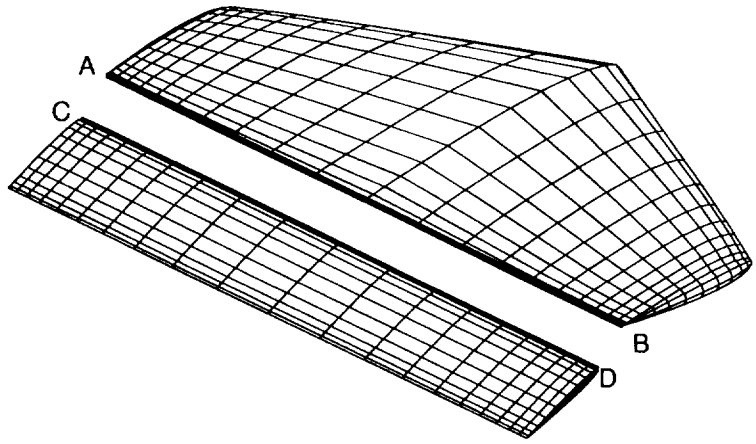
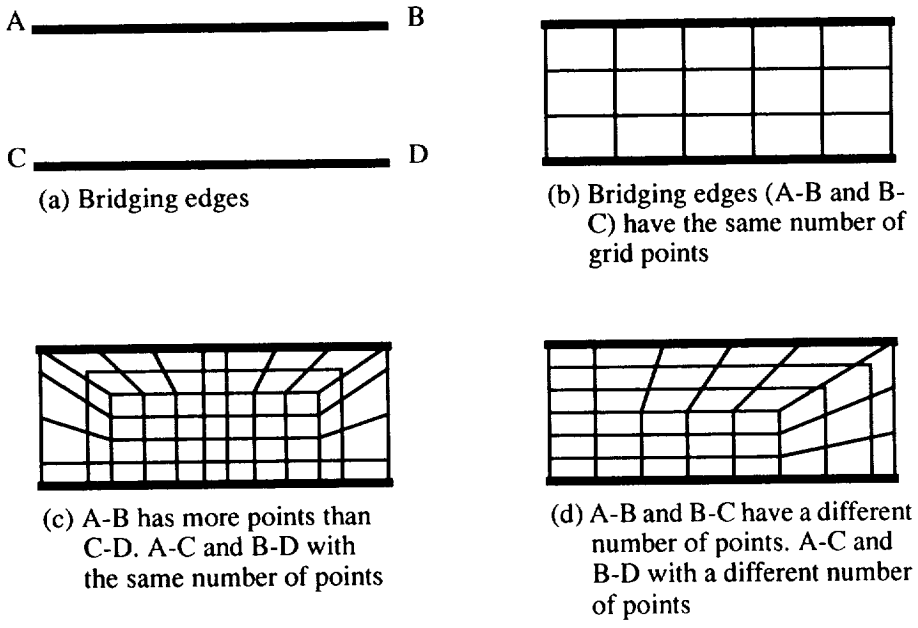



Fig. 5 A generic high-lift configuration wing and bridging edges (A-B and C-D)



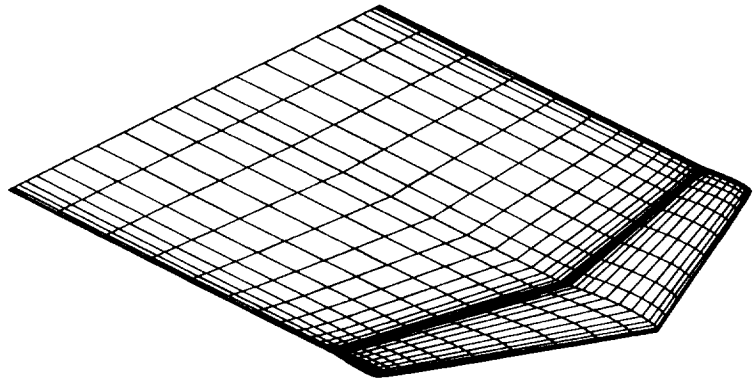

(a) Bridging edges

(b) Bridging edges (A-B and B-C) have the same number of grid points

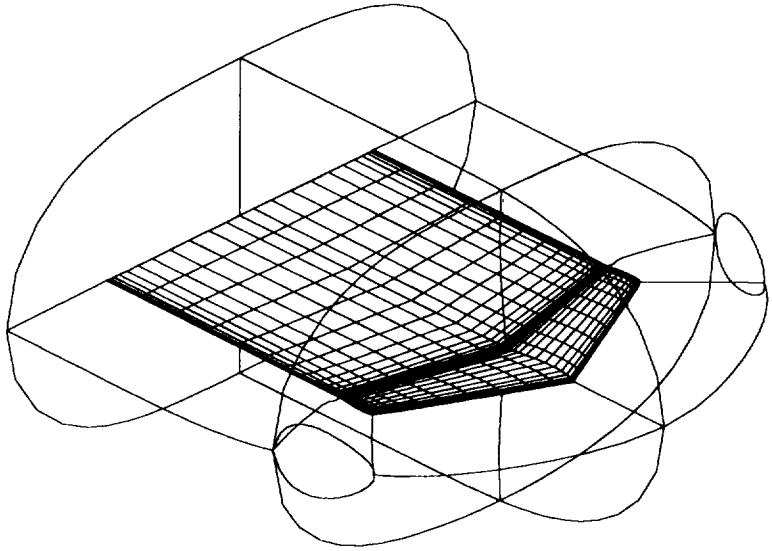
(c) A-B has more points than C-D. A-C and B-D with the same number of points

(d) A-B and B-C have a different number of points. A-C and B-D with a different number of points

Fig. 6 Formation of "bridging" fronts

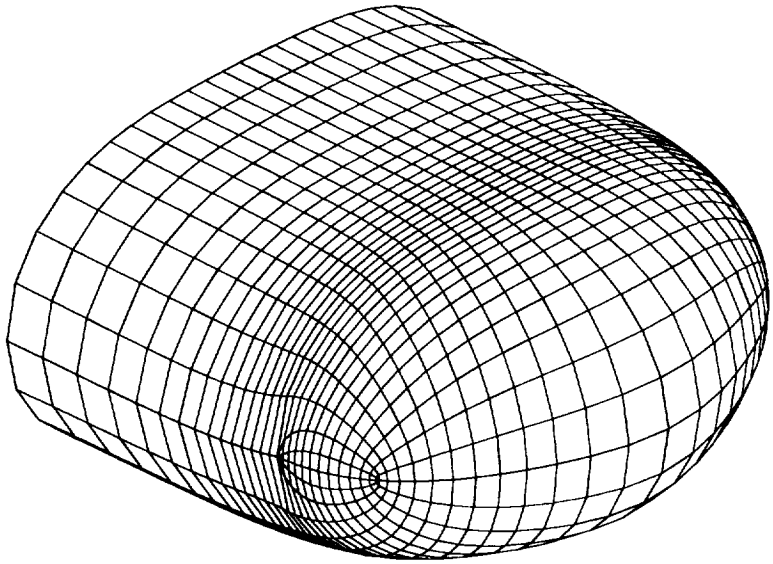


(a) Single initial front (body surface and wake plane)

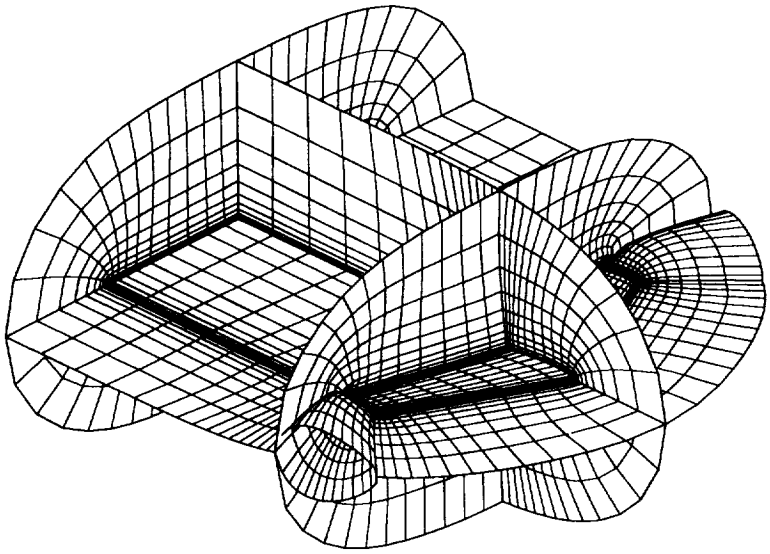


(b) Block edges of the resulting 6 block volume grid together with the boundary grid lines along the symmetry plane

Fig. 7 Grid generation for a single swept-back wing with a single initial front

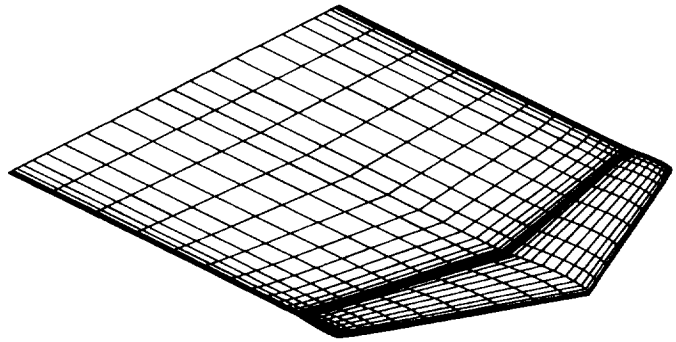


(c) Outer block boundaries

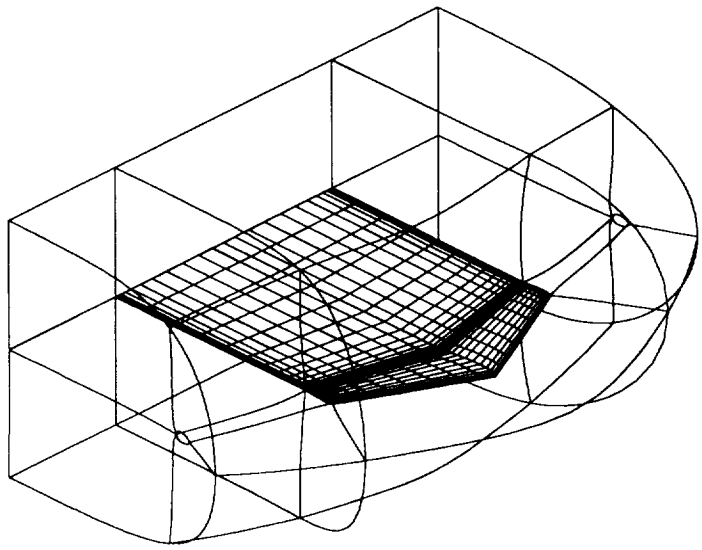


(d) Internal block boundaries and the symmetry plane

Fig. 7 (Continued) Grid generation for a single swept-back wing with a single initial front

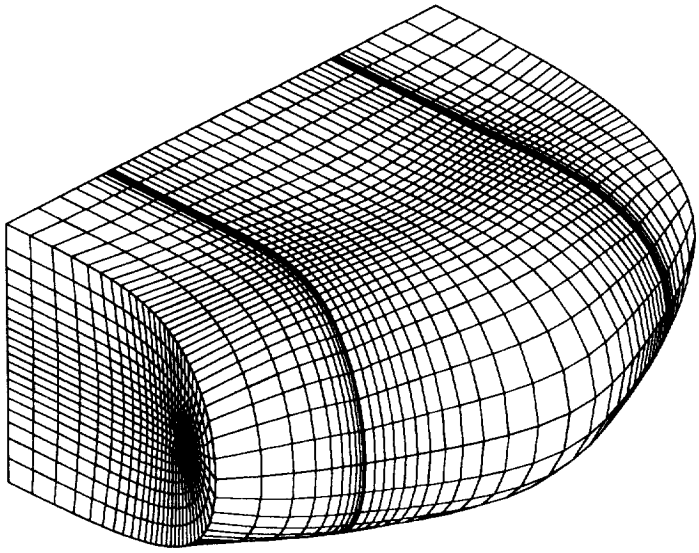


(a) 3 initial fronts (body surface and wake plane)

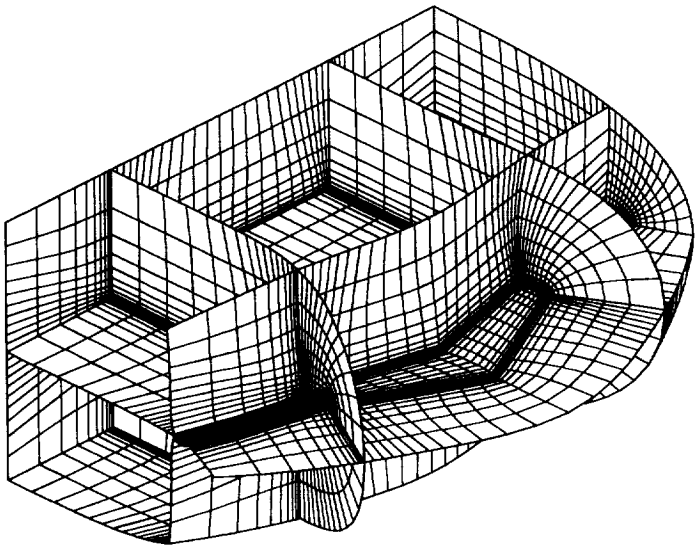


(b) Block edges of the resulting 14 block volume grid

Fig. 8 Grid generation for a single swept-back wing with 3 initial fronts

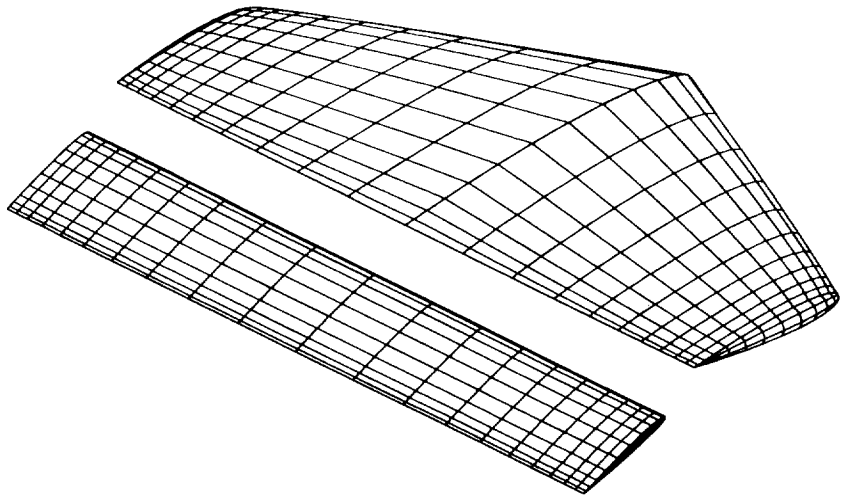


(c) Outer block boundaries

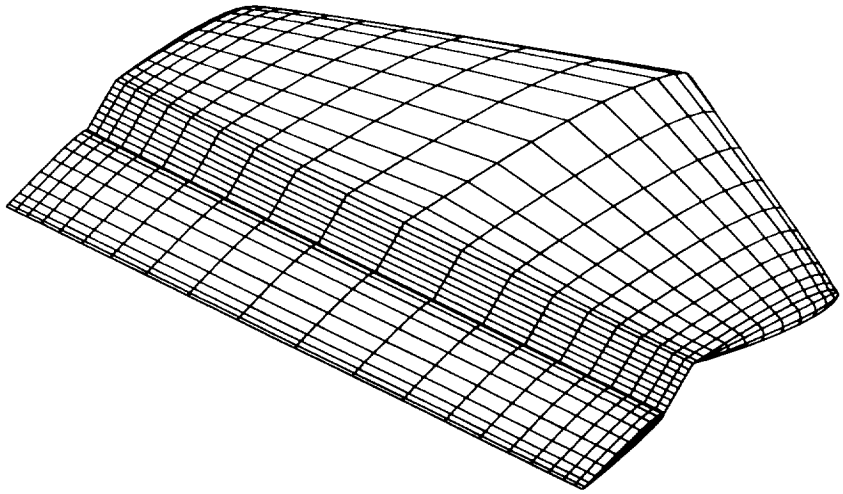


(d) Internal block boundaries

Fig. 8 (Continued) Grid generation for a single swept-back wing with 3 initial fronts

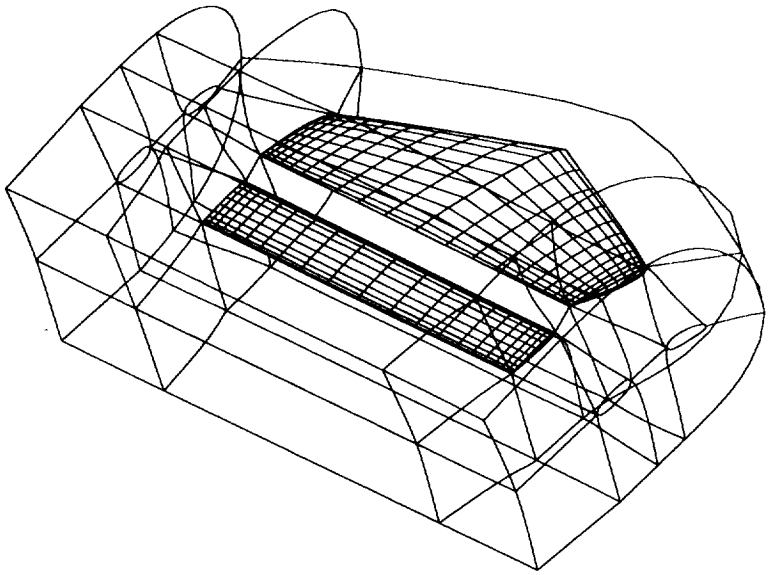


(a) Initial fronts (main wing and flap)

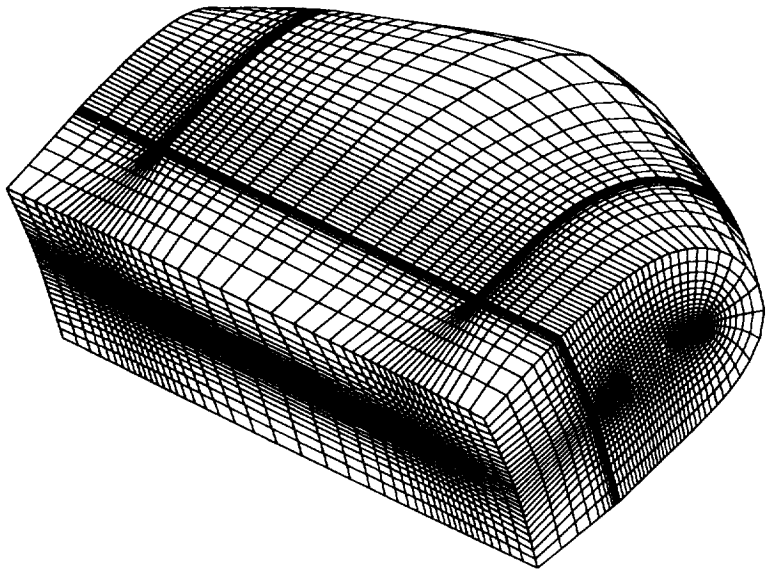


(b) Gap between the main wing and the flap is filled with a "bridging" front

Fig. 9 Grid generation for a generic high-lift configuration (28 block volume grid)

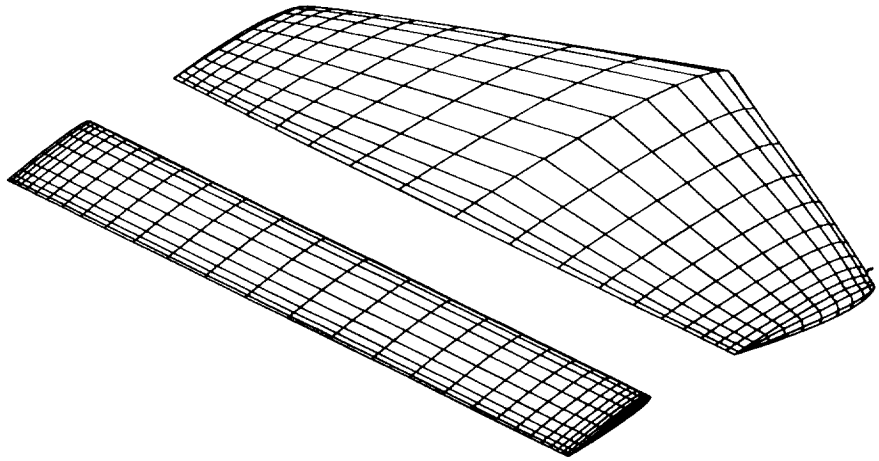


(c) Block edges of the resulting volume grid

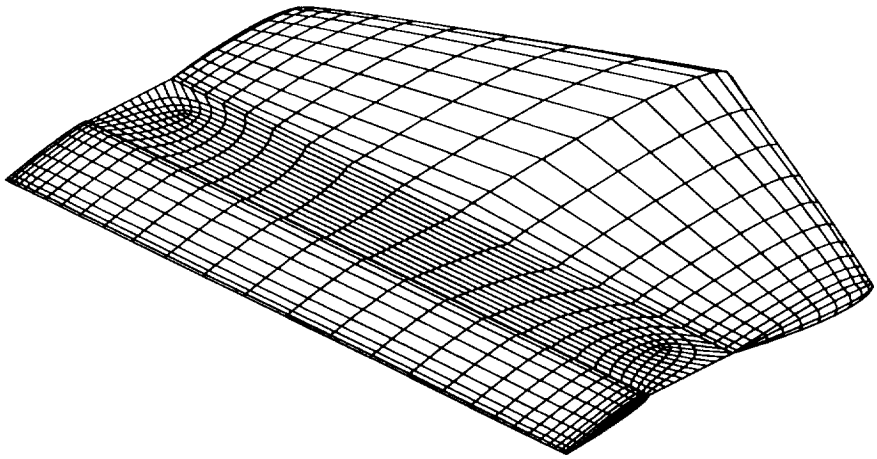


(d) Outer block boundaries

Fig. 9 (Continued) Grid generation for a generic high-lift configuration (28 block volume grid)

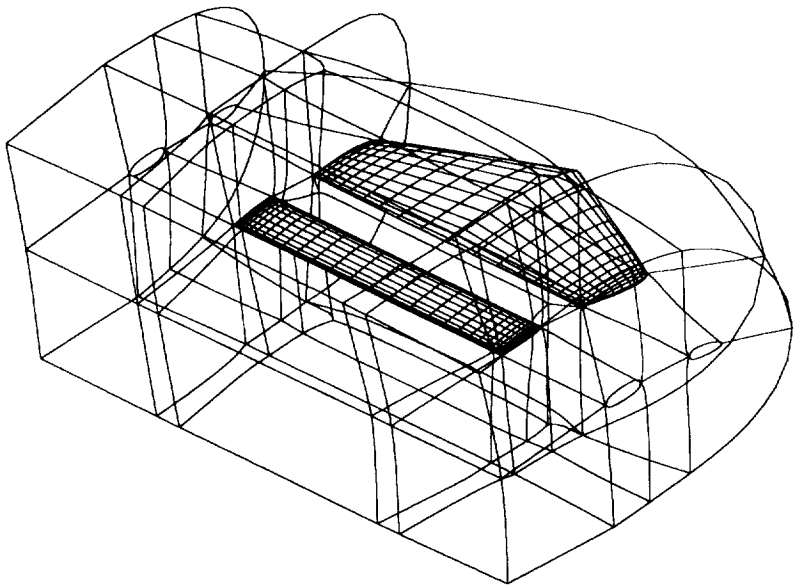


(a) Initial fronts (main wing and flap)

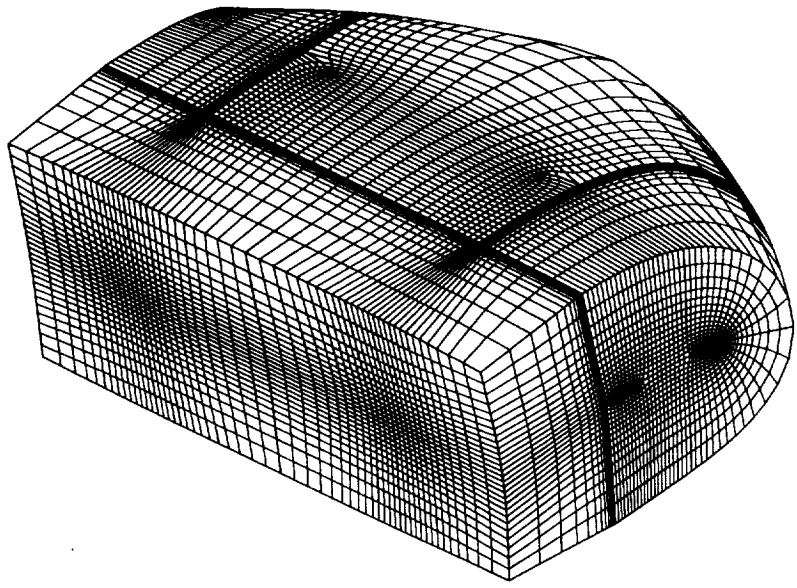


(b) Gap between the main wing and the flap is filled with a "bridging" front

Fig. 10 Grid generation for a generic high-lift configuration (38 block volume grid)



(c) Block edges of the resulting volume grid



(d) Outer block boundaries

Fig. 10 (Continued) Grid generation for a generic high-lift configuration (38 block volume grid)

N95- 28765

# MULTILEVEL ELLIPTIC SMOOTHING OF LARGE THREE-DIMENSIONAL GRIDS

C. Wayne Mastin*
Mail Stop 125
NASA Langley Research Center
Hampton, VA 23681

## SUMMARY

Elliptic grid generation methods have been used for many years to smooth and improve grids generated by algebraic interpolation schemes. However, the elliptic system that must be solved is nonlinear and convergence is generally very slow for large grids. In an attempt to make elliptic methods practical for large three-dimensional grids, a two-stage implementation is developed where the overall grid point locations are set using a coarse grid generated by the elliptic system. The coarse grid is then interpolated to generate a finer grid which is smoothed using only a few iterations of the elliptic system.

## INTRODUCTION

Elliptic grid generation methods have become less applicable to large scale problems due to the time required to solve the elliptic system of partial differential equations. The equations themselves are nonlinear and are difficult to solve efficiently even using the traditional multigrid methods. Although there are some variations of the equations, this report assumes the elliptic system is of the form

$$g^{11}(r_{\xi\xi} + P\ r_\xi) + g^{22}(r_{\eta\eta} + Q\ r_\eta) + g^{33}(r_{\zeta\zeta} + R\ r_\zeta) + g^{12}r_{\xi\eta} + g^{13}r_{\xi\zeta} + g^{23}r_{\eta\zeta} = 0$$

where $r = (x, y, z)$, $g^{ij}$ are the contravariant metric tensor components and the functions $P$, $Q$, and $R$ are used to control the distribution of grid points. The objective of this report is to demonstrate that in many cases it is not necessary to solve the elliptic system to generate a smooth grid with the required grid point distributions. If a coarse grid is first generated by solving the elliptic system, then this grid can be interpolated to generate a finer grid and the fine grid can be smoothed with only a few iterations of the elliptic difference equations. If this procedure is to work in practice, it is essential that the interpolated grid be smooth and give a good approximation of the final solution of the elliptic system on the fine grid. Thus, the fine grid iterations are primarily used to eliminate interpolation errors which are local and of high frequency. The actual residual on the fine grid may not be close to zero. This multilevel approach is efficient if only a few fine grid iterations are to give a smooth grid. It is well known that if the initial grid deviates greatly from the final elliptic grid, the first few iterations may generate large scale oscillations which decay very slowly. The multilevel approach can also be used to generate grids with specified boundary orthogonality and spacing. Coarse grid computations can be used to generate good initial approximations of control functions which may be fixed or further adjusted during the fine grid iterations.

*This work was performed while the author held a National Research Council - NASA Langley Research Center Research Associateship.

The techniques described in this report can be easily implemented in existing software packages. Coarse grids solutions have been used to generate starting values for the elliptic system in the GRIDGEN code of Steinbrenner and Chawner (ref. 1) and the 3DGRAPE code of Sorenson (ref. 2). Eisman (ref. 3) has also employed a similar concept in using elliptic systems to distribute control points for his algebraic grid generation scheme in the the GRIDPRO code.

It should be noted that the proposed scheme is not a true multigrid scheme. The scheme here is a simple grid sequencing scheme progressing from a coarse grid to progressively finer grids. There is no cycling between coarse and fine grids with the objective of eventually achieving a higher rate of convergence to the solution of the elliptic system of partial differential equations on the finest grid. Multigrid methods have been shown to increase convergence rates, but there have been no applications to large systems with moving boundary points and adaptive control functions. Some difficulties that may arise are discussed in the paper by Stüben and Linden (ref. 4).

## COARSE GRID ITERATIONS

The first step in the procedure is to coarsen the initial algebraic grid. This scheme that has been used here is to remove every other grid point in each coordinate direction. This refinement may proceed to several levels as long as the grid dimensions in each direction are odd integers. For large grids, this reduces the size of the problem by approximately eight. A second coarsening would reduce the original problem by a factor of approximately 64. Complementing the reduction in problem size is an increase in the rate of convergence when going to a coarser grid. The coarse grid iterations should be optimized for rapid convergence. For example, all coarse grid calculations here have used locally optimal acceleration parameters. Any control function and boundary condition options should be implemented so that the converged coarse grid has all of the desired characteristics of the final grid.

## INTERPOLATION

The interpolation scheme is a critical component in this procedure. Therefore, a tricubic Hermite interpolation procedure has been developed to generate a smooth grid. The slope information is calculated using central differences on the coarse grid. There are two options for the bounding surface grids. Either the original surface grids can be used or the surface grids can be redefined using the same interpolation scheme used at the interior points. The choice can have a significant effect on the success of the coarse grid solution in reducing the amount of work needed to obtain a usable grid. If the coarse grid accurately resolves the surface so that there is little change is curvature or spacings between grid points, then one can generally use the original fine surface grids and still have a reasonably smooth grid to start the elliptic system on the fine grid. On the other hand, if there are significant changes in surface or grid properties between the interpolated grid and the original surface grids, then using the original grids on the boundary surfaces will result in large changes in grid spacings and angles at the boundary, and possibly even some places where the interpolated grid folds over the boundary. This would result in a poor starting grid for the elliptic system and the main purpose in using the coarse grid solution would be lost.

The ability to prescribe grid distributions on the final grid is very dependent on the control functions of the elliptic system. There are two options for calculating control functions which seem

to work equally well. The control functions can either be calculated based on the initial fine grid and then restricted to each coarser grid, or the control functions can be calculated on each grid from restrictions of the initial grid to that level. It is generally recommended that the control functions not be computed on the coarsest grid and interpolated to the finer grids. This will often result in a loss of distribution on the finer grid. The only time that control functions have been interpolated to finer girds is when using the control functions to control boundary spacing an orthogonality. Even then, the interpolated control functions are blended with the control functions from the initial grid so that the interpolated values are only effective near the boundary.

There is one important fact that should be emphasized when transferring control functions between fine and coarse grids. When the control functions are restricted to a coarser grid, then they should also be multiplied by a factor of two. This scaling factor is necessary so that the same elliptic system is approximated on both the coarse and fine grids. Conversely, if the control functions are transferred from a coarse to a finer grid, then the control functions should be interpolated and divided by two. This assumes that the grid coarsening is done by removing every other point in each coordinated direction. Other coarsening schemes would result in different scaling factors. For example, if only every third point was retained in generating the coarse grid, then the factor of two would be replaced by three.

## FINE GRID ITERATIONS

At this point it is assume that the coarse grid iterations and interpolation procedure has resulted in a smooth grid that has the desired distribution of grid points. In many cases this grid would be good enough to compute a CFD solution. However, there may be a few ripples in the grid due to the Hermite interpolation. If the original boundary surfaces are maintained, the grid may need some additional smoothing near the boundary. Since the objective here is only to smooth the grid and not to obtain convergence of the elliptic system, there should be a change in the relaxation parameter so that the iterations are underrelaxed. Another effective way of smoothing the grid while maintaining the existing distribution of grid points is to introduce a time derivative into the partial differential equations, and solve the resulting parabolic system using a time marching method.

## EXAMPLES

Three sample grids will be considered to test the concepts of this report. All three initial grids were generated using transfinite interpolation. All three also are obviously not suited for CFD computations because of negative Jacobians or extremely skewed cells. After applying the elliptic smoothing, in two of the three cases the final grid was free of negative Jacobians. In the other case, a few negative Jacobians remained after using the elliptic methods, even when convergence of the elliptic system was attempted on the original fine grid.

The first example is the grid in the interior of a duct. The duct is plotted in Figure 1(a). The cross section is very irregular. An initial grid was constructed with dimensions of 33 by 33 by 65. There are a large number of grid points which fall outside of the duct as can be seem in the plot of an interior grid surface in Figure 1(b). This grid can be improved using elliptic methods, so that no negative Jacobians appear. In fact, negative Jacobians can be eliminated without resorting to

grid coarsening in only seven iterations. However, it takes many more iterations to give a smooth grid. The negative Jacobians in the initial grid cause large values of the residual in the solution of the elliptic system. Nonelliptic phenomena like ripples and waves can develop in the numerical solution. These perturbations may be of large magnitude and decay very slowly during the iterative solution. For example, the plot in Figure 1(c) is the same interior grid surface after ten iterations of the elliptic system. Ripples in the grid appear as triangular shaped cells in the lower part of the surface grid. Now consider the case where the original grid is coarsened to give an 17 by 17 by 33 grid. The elliptic system can be converged on this grid in less than sixty iterations, which is equivalent to less that eight iterations on the final grid. This coarse grid is interpolated and further smoothed to eliminate any rough spots near the boundary. For comparison with Figure 1(c), ten iterations on the fine grid was performed after convergence on the coarse grid, and the resulting grid plotted in Figure 1(d). As a further comparison, the original grid was used to calculate a converged solution of the elliptic system and it took about twice as many iterations to converge on the fine grid as it did on the coarse grid. Thus, when considering both the size of the grid and the convergence rate, it can be concluded that the use of coarse grid iterations resulted in a reduction in work by a factor of five over what would be needed to generate a converged elliptically generated grid. Zero control functions were used in all of these calculations. The grid points were fixed on the walls of the duct, but were allowed to slide along the two circular end caps.

The second example is a grid for a region about an aircraft wing. The initial grid is a C-grid constructed using transfinite interpolation. The grid was constructed as a four block grid each of which was 33 by 53 by 33 for a total of slightly over 230 thousand grid points. The edges of the blocks are illustrated in Figure 2(a) There are no negative Jacobians in this grid, but the grid is highly skewed in regions near the wing tip as can be seen in Figure 2(b). Several options have been exercised in this example which tend to reduce the rate of convergence for the elliptic system. The control functions have been interpolated from the block boundaries (using the Thomas - Middlecoff technique) to maintain the interior grid point distribution during the solution of the elliptic system. The control functions have been allowed to adjust near the surface of the wing (as in the GRAPE code) to generate orthogonal grid lines at the wing surface. Finally, the grip points have been allowed to float along the two planar boundary surfaces intersecting the ends of the wing. An indication of the slow rate of convergence is evident in Figure 2(c). After 10 iteration, starting with the original algebraic grid, there is little difference between the grid generated using the elliptic difference equations and the original algebraic grid. There is some indication of boundary orthogonality. However, there is considerable difference when examining Figure 2(d) which was generating using 100 coarse grid iterations followed by 10 fine grid iterations. The control functions were treated as described above. The control functions were first interpolated from the coarse grid to the fine grid and divided by two, since the coarse grid was generated by taking every other point of the fine grid. These control functions were then blended with the control functions computed from the initial interpolated grid. For this example, the original surfaces were used with the fine grid. It was therefore necessary to continue to adjust the control functions during the fine grid iterations to correct the slight skewness at the boundary which resulted from the interpolation.

The final example is included to demonstrate that this method can be applied to a grid with highly nonuniform spacing. The initial grid was again constructed using transfinite interpolation. The grid was to be used to compute viscous flow about an HSCT configuration. The wing/fuselage configuration is plotted in Figure 3(a). Since the HSCT configuration was symmetric, only half of the body was used to generate the grid. The grid was constructed in two blocks, each of which was 177 by 81 by 61. Thus the total grid consists of nearly one and three-quarter million points.

There are a large number of negative Jacobians. A comparison of the initial grid and the elliptic grid after 100 coarse grid iterations appears in Figures 3(b) and 3(c). For clarity, only the coarse grid points are plotted. The elliptic method was able remove most of the negative Jacobians and still maintain the spacing at the boundary. However, it could not remove all negative Jacobians and generate a grid which would be suitable for CFD calculations. While this may be considered a failure of elliptic methods, it is a successful application of the multilevel method. Knowing when and where elliptic methods fail to generate a suitable grid will allow the user to proceed with his efforts in redefining the topology or redistributing points.

## CONCLUSIONS

The discussion and examples contained in this report should give the grid generator a guide for using coarse grid iterations for smoothing and improving computational grids for CFD applications. The main point is that the coarse grid solution should be converged to some specified tolerance. After that, only a few fine grid iterations are needed. It is also important to treat the control functions correctly to generate the desired distribution of grid points along boundary surfaces. Simply interpolating these functions from the coarse grid to the fine grid is generally not sufficient.

There is one area where further study is needed. There should be some way of projecting an interpolated volume grid onto the original boundary surfaces without effecting the overall smoothness and orthogonality of the grid. One possible approach would be to include boundary information into the interpolation scheme.

## REFERENCES


1. Steinbrenner, J. P.; and Chawner, J. R.: " Recent Enhancements to the GRIDGEN Structured Grid Generation System," in *Software Systems for Surface Modeling and Grid Generation*, NASA CP-3143, NASA Langley Research Center, April 1992, pp. 253-271.

2. Sorenson, R. L.: *The 3DGRAPE Book: Theory, Users' Manual, Examples*, NASA TM-102224, NASA Ames Research Center, July, 1989.

3. Eisman, P. R.; Lu, N.; Jiang, M.; and Thompson, J. F.: "Algebraic - Elliptic Grid Generation", in *Numerical Grid Generation in Computational Fluid Dynamics and Related Fields*, Weatherill, N. P.; Eisman, P. R.; Häuser, J.; and Thompson, J. F. (eds.), Pineridge Press, Swansea, 1994, pp. 37-48.

4. Stüben, K.; and Linden, J.: "Multigrid Methods: An Overview with Emphasis on Grid Generation Processes", in *Numerical Grid Generation in Computational Fluid Dynamics*, Häuser, J.; and Taylor, C. (eds.), Pineridge Press, Swansea, 1986, pp. 483-509.

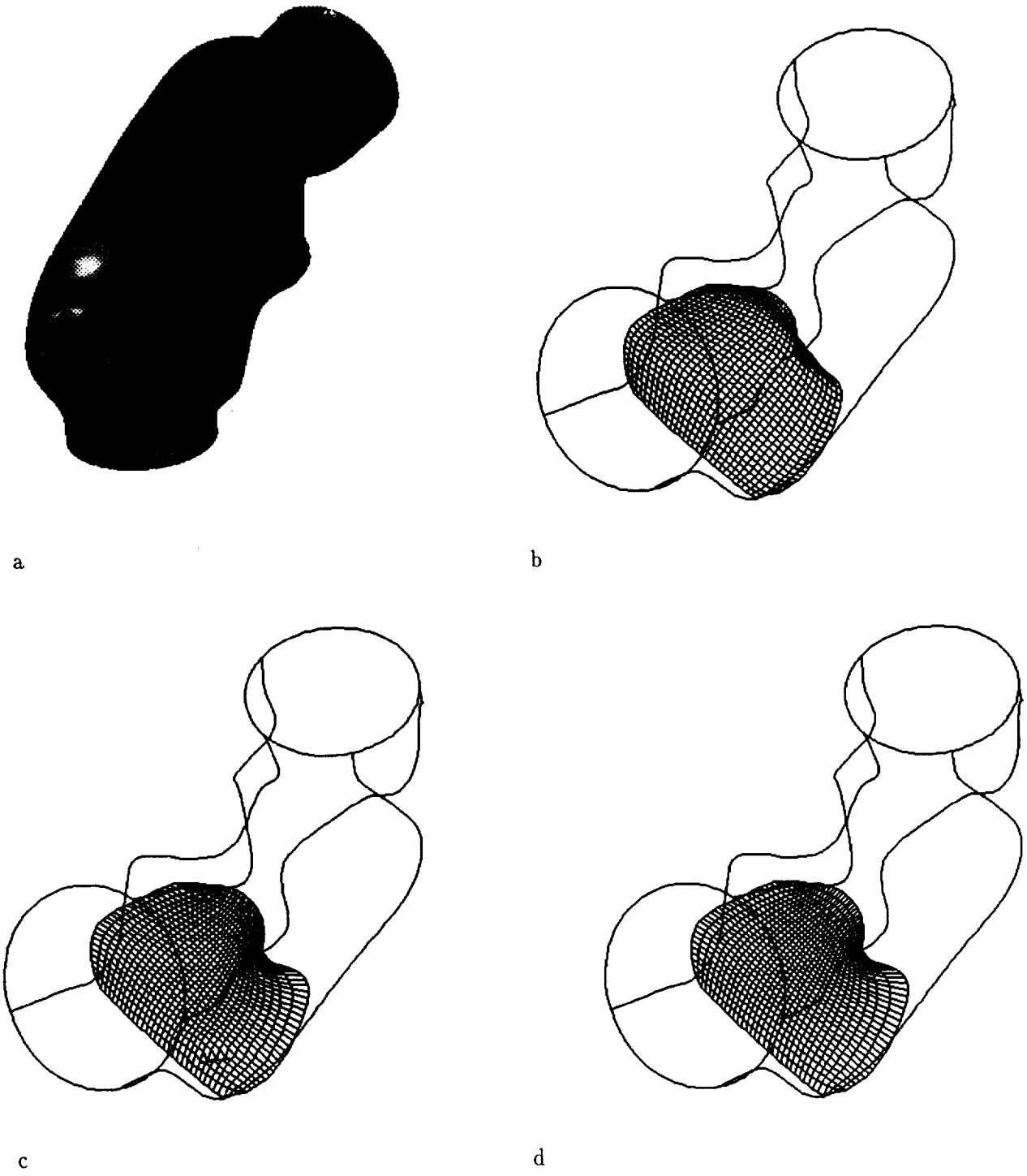



Figure 1. (a) Duct geometry, (b) initial surface grid, and surface grids after 10 iterations starting with (c) initial grid and (d) converged coarse grid solution.

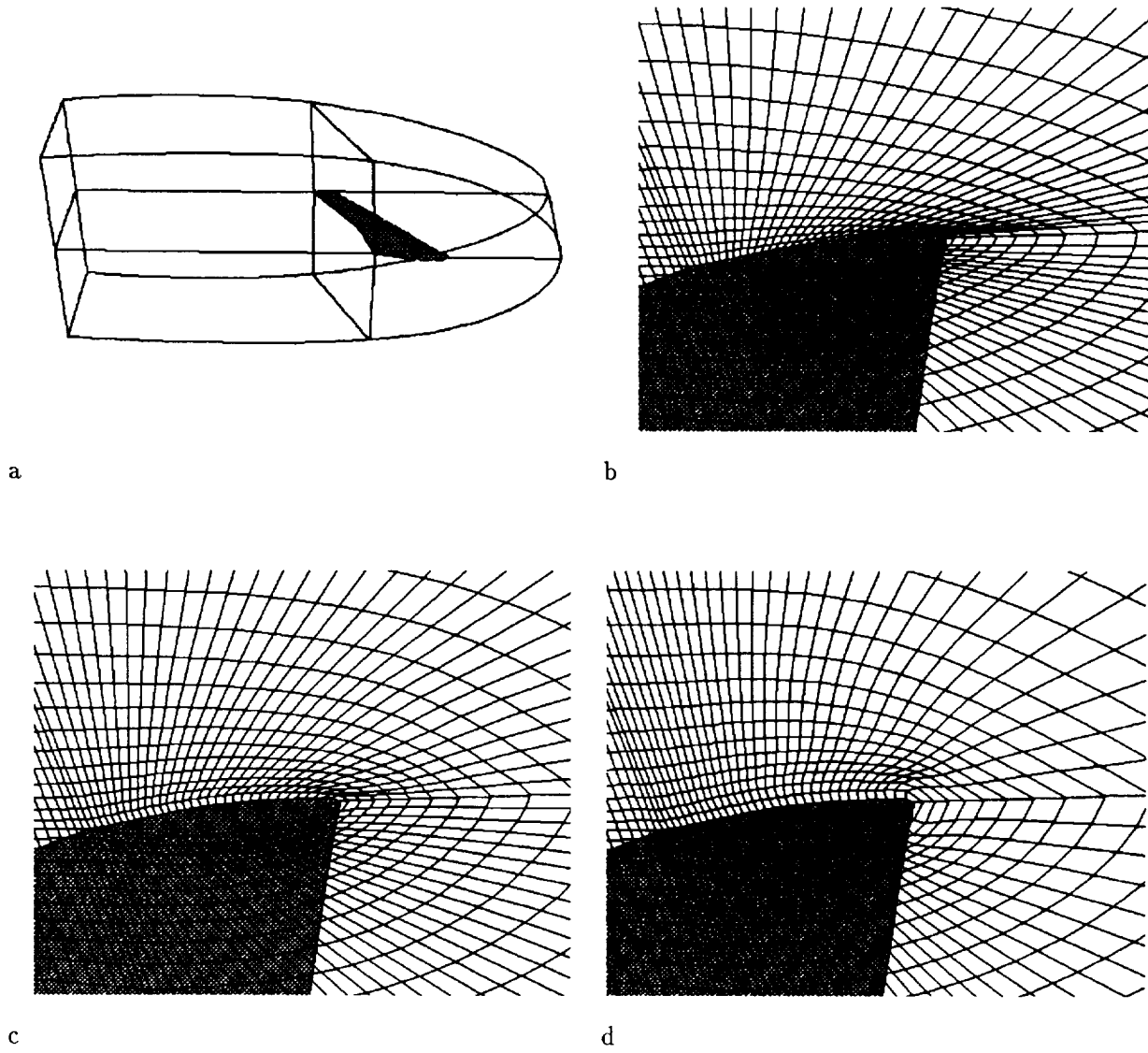
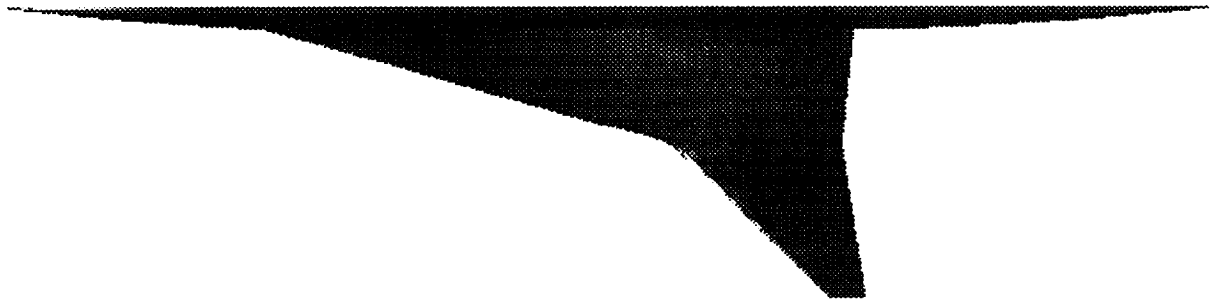
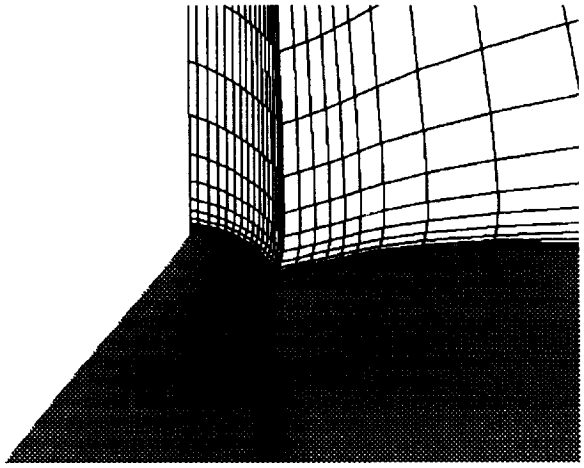


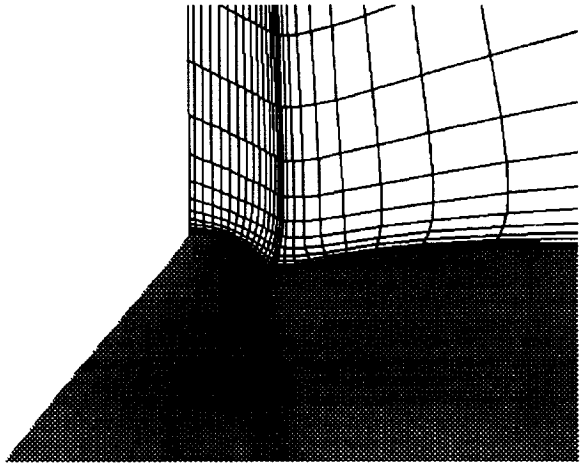

Figure 2. (a) Region about wing, (b) initial surface grid at wing tip, and surface grids after 10 iterations starting with (c) initial grid and (d) converged coarse grid solution.



a



b



c

Figure 3. (a) HSCT configuration, and (b) initial and (c) elliptic coarse grid at wing fuselage junction.

N95- 28766

# ADVANCED METHODOLOGY FOR SIMULATION OF COMPLEX FLOWS USING STRUCTURED GRID SYSTEMS

Erlendur Steinthorsson and David Modiano
Institute for Computational Mechanics in Propulsion,
NASA Lewis Research Center, Cleveland, Ohio

## ABSTRACT

Detailed simulations of viscous flows in complicated geometries pose a significant challenge to current capabilities of Computational Fluid Dynamics (CFD). To enable routine application of CFD to this class of problems, advanced methodologies are required that employ (a) automated grid generation, (b) adaptivity, (c) accurate discretizations and efficient solvers, and (d) advanced software techniques. Each of these ingredients contributes to increased accuracy, efficiency (in terms of human effort and computer time), and/or reliability of CFD software. In the long run, methodologies employing structured grid systems will remain a viable choice for routine simulation of flows in complex geometries only if genuinely automatic grid generation techniques for structured grids can be developed and if adaptivity is employed more routinely. More research in both these areas is urgently needed.

## INTRODUCTION

Computer technology and algorithms for Computational Fluid Dynamics (CFD) have developed to a point where CFD is now used routinely in the study of engineering and prototype flows. The purpose of using CFD is to give concrete predictions of specific data, such as surface pressure distributions or heat transfer, and to enhance understanding of the characteristics of the flows. Unfortunately, current methodologies and software lack the desired accuracy and efficiency for routine application to complicated viscous flows in complex geometries such as internal cooling passages of turbine blades. For these flows, large human effort and computing time are typically required to arrive at acceptable solutions. When structured grids are used in simulations, the process of arriving at a final solution often involves first, a considerable effort in generating an initial grid system and, thereafter, an iterative process involving obtaining a solution on the given grid, evaluating the solution, and refining or otherwise modifying the grid in preparation for computing another solution to the same problem.

Because of the large human effort needed in the past to generate high quality structured grids, many, whose interest is in the simulation of flows in complex geometries, have looked towards unstructured grids (mainly triangles in two dimensions, prisms and tetrahedra in three dimensions) to model the complicated geometries. Unstructured grids offer all the flexibility needed to model complicated geometries, and the grid generation process for unstructured grids is far more automated than that for structured grids. Also, algorithms for adapting unstructured grids to computed solu-

tions, for example through local mesh refinement, are well developed (see e.g., Ref 1). These advantages have led many to believe that unstructured grids are the best for use in simulations of flows in complex geometries. However, unstructured grids also have disadvantages, such as large memory requirements and/or large computing time (CPU time) requirements. Additionally, generating high-quality unstructured grids for high Reynolds number flows has proven difficult although recent progress has been made (see e.g., Ref. 2-4).

Due to difficulties like those mentioned above, high-fidelity simulations of viscous flows in complex geometries still pose a considerable challenge whether structured grids or unstructured are used. If the difficulties can be overcome through improved software and solution methodology, one can use existing computing resources to tackle bigger and more complicated problems than can now be handled, and make more routine the simulation of flows now analyzed only with great difficulty. For organizations employing CFD, this translates into economic savings through better use of manpower and computing equipment. In this paper, an advanced CFD methodology is outlined that will overcome at least some of the difficulties related to the use of structured grids. The ideas behind the methodology are not necessarily new but they are rarely put forward in a systematic way as done here. In the next section, the ingredients in methodology are discussed in general terms that apply independent of grid structure. Following the general description, the role of grid structure is discussed and a specific strategy for structured grids is laid out. The paper ends with a short summary.

## METHODOLOGY

The thesis of this paper is that in order to optimize the use of human and computing resources, any methodology for simulating flows in complex geometries needs certain "ingredients." These ingredients are

(a) automated grid generation,

(b) adaptivity,

(c) accurate discretizations of the governing equations and efficient solvers,

(d) advanced software techniques.

All these ingredients or components contribute to the effectiveness and reliability of the software used for flow simulation. The importance of each is discussed below. A diagram showing the impact of integrating the components in a single methodology is shown in Fig. 1.

Automated grid generation is now recognized as an important step towards an effective simulation methodology. It reduces the human effort needed to generate grids for complicated geometries and can significantly cut the total time needed to arrive at acceptable solutions. In the past, the time needed to generate a structured grid for a complicated geometry was often an order of magnitude greater than the time needed to compute a solution. Today, this may still be true although

new grid generation methods and improved graphical user interface with the grid generation software have remedied this problem somewhat. Grid generation methods for unstructured grids are typically much more automated than for structured grids. For instance, advancing front techniques for generation of triangular and tetrahedra meshes allow grids for very complicated geometries to be generated quickly (in a matter of hours or even minutes) with little human intervention.[5-7] Generation of unstructured Cartesian grids is even more automated and can be completed in minutes.[8]

Adaptivity—i.e., automatically adjusting mesh resolution (h-refinement) or order of accuracy of the discretization (p-refinement) in response to the computed solution—can also bring important improvements to the overall solution process. The improvements include enhanced accuracy of the computed solutions, reduced computational effort, and reduced human effort. Accuracy is enhanced through improved resolution of important flow features, even when it is not known beforehand where these flow features are located. Computational effort is reduced as adaptivity allows appropriate resolution to be used everywhere in the flow field, e.g., adaptive local mesh refinement produces fine grids only where they are needed leaving appropriately coarser grids elsewhere. Possibly the greatest benefit of adaptivity is the reduction or even elimination of the iterative process that typifies the present use of CFD for complex flows, where a user, iteratively, computes a solution on a given grid, evaluates the solution with respect to accuracy, and then modifies the grid "manually" in preparation for computing a new solution to the same problem. Adaptivity takes on added importance with increasing demands for accurate solutions and for concrete error estimation for the computed solutions. The increased demands for accuracy are evident, for example, in new editorial policies by professional journals such as the Journal of Fluids Engineering.[9]

The need for good discretizations and efficient solvers for flows in complex geometries is self explanatory in the light of the large number of grid points needed for high fidelity simulations of such flows. However, it is important to note that memory requirements of the flow solvers is also an issue for problems involving a large number of grid points. The memory required by a flow solver is often what limits the size of the problems that can realistically be tackled using that particular flow solver on a particular computer.

The need for advanced software techniques arises inevitably as the increased automation of the solution procedures and increased use of advanced solution algorithms makes the CFD software more and more complicated. Increased generality of the software and improved user interface only add to the complexity. Consequently, the software must be carefully and systematically developed to ensure bug-free codes and reliable operation. Advanced software techniques, such as object oriented programming is one way to achieve this. Modern programming languages that support object oriented programming include C++, which supports object oriented programming without sacrificing efficiency.[10] The new FORTRAN90 programming language also has some support for object oriented program design. Object oriented programming techniques enhance the modularity, reliability and

maintainability of computer codes, and contribute to the development of bug-free modules. They are already in use for development of CFD codes (see e.g., Ref. 11, 12).

When all are combined, the ingredients discussed above can significantly enhance the capabilities of CFD software and widen the range of applications that can be tackled on a given computing platform. The results that are obtained will also be more reliable and the power of CFD can more easily be exploited by users who are not experts in CFD or fluid mechanics.

## STRUCTURED GRIDS FOR COMPLICATED GEOMETRIES

To a varying degree of sophistication, current CFD software makes use of some or all of the ingredients of the ideal methodology described in the previous section. With respect to automation of the grid generation and adaptivity, CFD methodologies employing unstructured grids come closest to the ideal. However, a persistent drawback of CFD codes employing unstructured grids is a large memory requirement. Due to lack of structure in the grids, information about connectivity between cells or nodes in the grid needs to be stored. Also coefficients related to reconstruction of the solution based on cell average values are stored in many codes. A total storage of about 180 double precision variables (over 1400 bytes) per solution element (e.g., cells in cell-centered schemes) appears to be typical for flow solvers capable of simulating three-dimensional viscous flows (see e.g., Ref. 13, 14). In comparison, codes employing structured grids typically require between 45 and 70 double precision variables per solution element. Thus simply from the perspective of storage requirements, structured grids are preferable to unstructured grids for problems requiring high resolution of the flow field since they allow larger problems to be tackled on a given computer.

In addition to requiring less memory, structured grids offer other advantages. First, due to the regularity of the grid structure, various efficient schemes based on dimensional splitting can be applied on structured grids. For instance, ADI schemes and line-Gauss-Seidel methods can be used effectively for implicit discretizations. Also due to the regularity of the grid structure, multigrid schemes can be applied in a straightforward manner on structured grids to accelerate convergence to steady state or to speed up the resolution of systems of equations arising from implicit discretizations. Finally, structured body fitted grids are well suited for simulations of viscous flows due to good resolution of boundary layer regions.

Based on the considerations touched upon above, structured grids appear at the present time to be better suited than unstructured grids for simulation of complex flows requiring high resolution of the flow field. However, three questions arise immediately: First, can efficient structured grids be generated for very complicated geometries? Second, can the generation of structured grids be automated to the extent needed? Finally, can adaptivity be implemented in an effective manner? If the answer to any of these questions is negative then structured grids will in the long run only

be useful for a very limited class of flow problems. The first two questions will be addressed in the remainder of this section. The last question will addressed in the next section.

### Efficient Structured Grids for Complicated geometries

The first question to be answered regarding the use of structured grids is whether *efficient* structured grids can be generated for very complicated geometries. This question is best addressed by posing it as two questions, namely for how complicated a geometry can a high quality structured grid be generated and when are the grids efficient?

A number of strategies have been devised to generate structured grids for complicated geometries. Two main approaches are to use "multiblock" grids (continuous block-structured body-fitted grid systems) and "Chimera" grids.[15] Multiblock grids are generated by "carving up" the flow domain into multiple zones or blocks of reasonable size and shape such that a continuous grid can be generated for the entire domain. In contrast, Chimera grids are created by independently generating simple body-fitted grids around different components of a geometry, allowing the component grids to overlap in an arbitrary manner. Multiblock grids have the greatest degree of continuity of all block structured grids while Chimera grids have the least. A number of alternative structures have been devised that, in terms of continuity across blocks boundaries, fall in between the two main structures.

The advantage of using Chimera grids is that such grids can be generated for almost any geometry. In fact, the flexibility of Chimera grids closely approaches that of unstructured grids. Furthermore, grid generation effort is greatly simplified since the geometry can be broken up into components for which simple, high quality grids can be generated. A disadvantage is that the flow solver must deal with the arbitrary overlap between grids. In a sense, the difficulty in dealing with the complicated geometry has been postponed from the grid generation stage to the flow-simulation stage. Often, the interpolation needed to transfer data between component grids leads to noticeable glitches in the computed solutions. Recently, new approaches have been proposed that promise to to improve the communication between grids. One is to replace the region of overlap with an unstructured grid (DRAGON grids, see Ref. 16). Another is to eliminate the overlap region from one of two overlapping grids and to enforce flux conservation at the surface (curve in two dimensions) of intersection between the two blocks.[17] Commercial grid generators such as ICEM-CFD[18] and Gridgen[19] are well suited for generating Chimera grids. A number of flow solvers are designed to use Chimera grids, including OVERFLOW (developed at NASA Ames Research Center) and CFL3D (developed at NASA Langley Research Center).

The advantage of multiblock grid systems is the high degree of continuity in the entire grid system. In high quality multiblock grids, grid lines are at least $C^1$ continuous across block boundaries (except at certain singular points in the grids). Consequently, implementation of schemes for multiblock grids in flow solvers is relatively straightforward and block-boundaries become transparent

to the solver. For instance, higher-order accurate finite volume discretizations can be implemented without loss in accuracy at block boundaries. The disadvantage of multiblock grids is that it has traditionally been considerably harder to generate high-quality multiblock grids than almost any other type of grid system. Recently, however, new methods[20-22] and commercial software[23] have become available that go a long way towards making multiblock grids competitive with Chimera grids, both in terms of human effort in grid generation and in terms of capability to handle complicated geometries. A sample grid multiblock grid system generated by the commercially available GridPro[23] is shown in Fig. 2 and 3. The strength of this particular software package is that the only input required, in addition to the geometry itself, is the topological structure of the grid system. Exact specification of block interfaces is not needed. Flow solvers designed to use multiblock grids include TLNS-MB[24] and TRAF3D.MB.[25]

Whether a structured grid for a complicated geometry will be efficient must be answered on a case by case basis since the answer depends not only on the geometry but also on the resolution that is required. To be efficient, each grid block should have at least few hundred nodes or cells (in three-dimensions). Otherwise, too much memory and computer-time overhead will be involved in communicating data between blocks. In general, if high-fidelity simulations of viscous flows are planned, the grid spacing required for accuracy of the solution will be orders of magnitude smaller than the physical size of the blocks used in a multiblock or Chimera grid system to resolve the features of the geometry. In most such cases then, efficient structured grids can be generated.

### Automation of Grid Generation

The second question to be answered regarding the use of structured grids was whether the grid generation can be automated. Unfortunately, up to this point little effort has gone into developing truly automatic algorithms for generation of structured grids. It is therefore premature to judge whether effective algorithms can be developed. However, some recent work has been published on automatic generation of multiblock grids for two-dimensional geometries. In particular, Shoenfeld and Weinerfelt[26] devised an algorithm based on an advancing front technique. This same technique can be used to generate an initial topology of a grid system while other techniques can be used to optimize the grid for that particular block structure. It therefore appears likely that automatic techniques can be developed that work for two-dimensional geometries. For three-dimensional geometries, other techniques based for instance on feature recognition may need to be used.

## MESH REFINEMENT IN STRUCTURED GRIDS

The third and last question related to the use of structured grids in simulations of complex flows was whether adaptivity could be effectively implemented for structured grids. As discussed earlier, the objective of using adaptivity in flow simulations is twofold. First, to ensure that without human

intervention all important features of the flow are captured to the desired level of accuracy, and second, to minimize the computational effort required to achieve the desired accuracy. Two approaches have been used to implement adaptivity in finite volume schemes. The first is to move grid points in the grid system towards regions where higher resolution is needed, leaving coarser grids elsewhere. The second approach is to locally refine the mesh where higher resolution is needed by adding points (local Adaptive Mesh Refinement or AMR). The moving mesh approach can be used to maximize the accuracy of a solution computed on a grid with a fixed number of nodes; in a sense, maximizing accuracy for a given cost. When used in structured grids it has the limitation that grid points can only move a certain amount before grid quality (smoothness and orthogonality) degrades and begins to counteract the advantages of increased resolution. Adaptive mesh refinement has no such limitation and multiple levels of refinement can be used to achieve the desired accuracy (within the limits of applied models such as turbulence models, etc.). Thus, AMR offers the needed flexibility to automate adaptivity in the solution process.

To be compatible with the use of structured grids, an AMR algorithm should allow the flow solver to take full advantage of the strengths of structured grid systems, including efficient solution algorithms for implicit discretizations, various schemes based on dimensional splitting, and multigrid schemes. In essence, this means that the algorithm should use a block structure for the refined grid. To date, only a few methodologies of this nature have been proposed. One of the earliest was the method of Berger and Oliger.[27] In their method, the refined grids are allowed to overlay the underlying coarse grids in an arbitrary manner. The blocks of the refined grids are constructed in physical space based on the shape and size of the region to be refined. The resulting grids tend to align with discontinuities and other features that determine the shape and size of the region.

Building on the work of Berger and Oliger, Berger and Colella[28] devised a methodology in which the refined grids conform with the coarse grids, i.e., the boundaries of the fine-grid blocks are made to coincide with grid lines of the coarse grid. In this method, the block structure for the refined grids is created using a special algorithm that fits topologically rectangular patches over the regions of the coarse grids where the estimated error in the solution is above a specified threshold value. The blocks of the refined grids are then created by subdividing the coarse grid cells within each of the patches. Advantages of the new approach include greatly simplified prolongation and restriction operators for transferring data between a coarse grid and a refined grid, and that conservation at interfaces between coarse and fine grids can be enforced in a rigorous manner. This approach has been used extensively for Cartesian grids (see e.g., Ref 29, 30). Efforts to implement the algorithm in structured grids, including Chimera grids, are ongoing.[31,32]

A third AMR algorithm, proposed by Davis and Dannenhoffer[33], also involves creating the fine grids by subdividing cells on the coarser grid. In this approach, however, the entire structured grid system is divided up into *a priori* defined sub-blocks of uniform size and dimensions. During adapta-

tion, each sub-block is refined either in its entirety or not at all. In a recursive manner, a sub-block that has been refined is itself divided into sub-blocks, each of which can be refined. In the algorithm of Davis and Dannenhoffer, directional refinement is used, i.e., the grid can be refined in one, two or three directions at a time as desired.

To demonstrate the feasibility of using AMR in structured grids, results obtained using AMR in simulation of a two-dimensional inviscid transonic flow over a NACA0012 airfoil are shown in Fig. 4. The AMR algorithm used is that of Berger and Colella.[27] The adaptation of the algorithm to structured grids is described in Ref. 31. The discretization used for the governing equations is also described in Ref. 31. The solution shown in Fig. 4 was computed using three levels of refinement with a refinement ratio of two. The dimensions of the original coarse grid around the NACA airfoil were $32 \times 96$. Only about 20% were refined three times (to the finest level). According to available data[1] the error in the solution is minimal, less than 2% as measured by the location of the normal shock behind the airfoil.

Based on the above review of AMR methods for structured grids, it is fair to say that the ability to use solution-adaptive mesh refinement to implement adaptivity in structured grids has been demonstrated. Nonetheless, further demonstrations may be needed, particularly for three-dimensional flows and for multiblock grid systems. Here it is appropriate to point out that implementing AMR in a general purpose flow solver requires careful programming and the use of a modern programming language. In fact, due to limitations of FORTRAN77, related to dynamic memory allocation and advanced data structures, it may be impossible to write in that language a practical general purpose flow solver that makes use of AMR. One can speculate that it is precisely because of the limitation of that otherwise effective programming language for scientific computing that use of AMR for structured grids is not already more wide spread.

## SUMMARY

In order to optimize the use of human and computing resources, any methodology for simulating flows in complex geometries needs the following ingredients: (a) automated grid generation,(b) adaptivity, (c) accurate discretizations and efficient solvers, and (d) advanced software techniques. For simulations of viscous flows in complicated geometries or other flows requiring high resolution of the flow field, structured grid systems offer clear advantages over unstructured grids due to less memory usage and efficient flow-solver technology. However, in terms of automation of grid generation and adaptivity, methodologies employing structured grids trail those using unstructured grids.

In the long run, structured grids will remain a viable choice for routine simulations of flows in complicated geometries only if sufficiently efficient automatic grid generation techniques for structured grids are developed and only if adaptivity is employed in the solution methodology. At the

present time, automatic grid generation methods for two-dimensional geometries may be on the horizon and adaptivity based on local adaptive mesh refinement schemes that are compatible with the use of structured grids have been demonstrated or are under development. More research is urgently needed in both areas.

## REFERENCES


1 Warren, G. P., Anderson, W. K., Thomas, J. L., and Krist, S. L.; "Grid Convergence for Adaptive Methods," AIAA 91-1592, June 1991.

2 Pirzadeh, S.; "Unstructured Viscous Grid Generation by the Advancing Layers Method," AIAA 93-3453, August, 1993.

3 Pirzadeh, S.; "Viscous Unstructured Three-Dimensional Grids by the Advancing Layers Method," AIAA 94-0417, August, 1993.

4 Ward, S. and Kallinderis, Y.; "Hybrid Prismatic/Tetrahedral Grid Generation for Complex 3-D Geometries," AIAA 93-00669, January, 1993.

5 Peraire, J.V., Morgan, M., and Zienkiewicz, O.C.; "Adaptive Remeshing for Compressible Flow Computations," *J. Comp. Phys.*, Vol. 72, October, 1987, pp. 449–466.

6 Gumbert, C., Lohner, R., Parikh, P. and Pirzadeh, S.; "A Package for Unstructured Grid Generation and Finite Element Flow Solvers," AIAA 89-2175, June, 1989.

7 Mavriplis, D. J.; "An Advancing Front Delaunay Triangulation Algorithm Designed for Robustness," ICASE Report, 92-49, October, 1992.

8 Coirier, W. J.; *An Adaptively-Refined Cartesian, Cell-Based Scheme for the Euler and Navier Stokes Equations*, PhD thesis, University of Michigan, Department of Aerospace Engineering, 1994. Also, NASA TM 106754.

9 "Journal of Fluids Engineering Editorial Policy Statement on the Control of Numerical Accuracy," *Transactions of the ASME, Journal of Fluids Engineering*, Vol. 115, Sept. 1993, p. 340.

10 Stroustrup, B.; "*The C++ Programming Language*," 2nd edition, Addison Wesley, 1991.

11 Crutchfield, W. Y. and Welcome, M. L.; "Object Oriented Implementation of Adaptive Mesh Refinement Algorithm," *Scientific Programming*, Vol. 2, number 4, winter 1993.

12 Tchon, K.-F.; "Object-Oriented Programming for a Temporal Adaptive Navier Stokes Solver," AIAA 95-0574

13 Frink, N. T.; "Recent Progress Toward a Three-Dimensional Unstructured Navier-Stokes Flow Solver," AIAA-94-0061.

14 Mavriplis, D.J., and Venkatakrishnan, V.; "A 3-D Agglomeration Multigrid Solver for the Reynolds-Averaged Navier Stokes Equations on Unstructured Meshes," AIAA 95-0345, January, 1995.

15 Steger, J.L., Dougherty, F.C. and Benec, J.A.; "A Chimera Grid Scheme," Advances in Grid Gener-

ation, Ed. by K.N. Ghia, and U. Ghia, 1983, pp. 59-70.

16 Liou, M.-S., and Kao, K.-H.; "Progress in Grid Generation: From Chimera to DRAGON Grids," *Frontiers of Computational Fluid Dynamics 1994*, (D.A. Caughey and M. M. Hafez, ed's), Chapter 20, John Wiley and Sons, Nov. 1994.

17 Wang, Z.J., Buning, P., and Benek, J.; "Critical Evaluation of Conservative and Non-Conservative Interface Treatment for Chimera Grids," AIAA 95-0077, January, 1995.

18 Akdag, D. and Wulf, A.; "Integrated Geometry and Grid Generation System for Complex Configurations," NASA CP 3143, April, 1992, pp. 161-171.

19 Steinbrenner, J.P. and Chawner, J.R.; "The Gridgen Version 9, Multiple-Block Grid Generation Software," MDA Engineering Report 94-01, June 1994.

20 Allwright, S. E.; "Techniques in Multiblock Domain Decomposition and Surface Grid Generation," in *Numerical Grid Generation for Computational Fluid Mechanics '88*; Sengupta, S., Häuser, J., Eiseman, P. R., and Thompson, J. F., (ed), pp. 559-568, Pineridge Press, 1988.

21 Amdahl, D. J.; "Interactive Multi-Block Grid Generation," in *Numerical Grid Generation for Computational Fluid Mechanics '88*; Sengupta, S., Häuser, J., Eiseman, P. R., and Thompson, J. F., (ed), pp. 579-578, Pineridge Press, 1988.

22 Stewart, M.; "A General Decomposition Algorithm Applied to Multi-Element Airfoil Grids," AIAA 90-1606, 1990.

23 "GRIDPRO,$^{TM}$ /AZ3000, Users Guide and Reference Manual", Program Development Corporation, White Plains, NY, 1993.

24 Vatsa, V. N., Sanetrik, M. D. and Parlette, E.B.; "Development of of a Flexible and Efficient Multigrid-Based Multiblock Flow Solver," AIAA 93-0677, January, 1993.

25 Steinthorsson, E., Liou, M.-S., and Povinelli, L.A.; "Development of an Explicit Multiblock / Multigrid Flow Solver for Viscous Flows in Complex Geometries,"
AIAA-93-2380, 1993. Also, NASA TM 106356 and ICOMP 93-34.

26 Schönfeld, T. and Weinerfelt, P.; "Automatic Generation of Quadrilateral Multi-Block Grids by the Advancing Front Technique," Proceedings of the *Third International Conference on Numerical Grid Generation*, pp. 743-754, S.-Arcilla, A., Häuser, J., Eisemann, P.R., and Thompson, J.F. (Eds.), Elsevier Science Publishers, North Holland, June 1991.

27 Berger, M. J. and Oliger, J.; "Adaptive Mesh Refinement for Hyperbolic Partial Differential Equations," *J. of Comp. Physics*, Vol. 53, pp. 484-512, 1984.

28 Berger, M. J. and Colella, P.; "Local Adaptive Mesh Refinement for Shock Hydrodynamics," *J. of Comp. Physics*, Vol. 82, pp. 64-84, 1989.

29 Bell, J. B., Berger, M. J., Saltzman, J. S., and Welcome, M. L.; "Three-Dimensional Adaptive Mesh Refinement of Hyperbolic Conservation Laws," UCRL-JC-108794, Lawrence Livermore Natioanl Laboratory, December, 1991.

30 Pember, R. B., Bell, J. B., Colella, P., Crutchfield, W. Y., and Welcome, M. L.; "Adaptive Cartesian Grid Methods for Representing Geometry in Inviscid Compressible Flow," UCRL-JC-113851, Lawrence Livermore Natioanl Laboratory, June, 1993.

31 Steinthorsson, E., Modiano, D. and Colella, P.; "Computations of Unsteady Viscous Flows Using Solution Adaptive Mesh Refinement in Curvilinear Body-Fitted Grid Systems" AIAA-94-2330, June, 1994. Also, ICOMP-94-17, NASA TM 106704.

32 Brislawn, K. D., Brown, D. L., Chesshire, G. S., and Saltzman, J. S.; "Adaptively Refined Overlapping Grids for the Numerical Solutions of Systems of Hyperbolic Conservation Laws," LA-UR-95-257, Los Alamos Natioanl Laboratory.

33 Davis, R. L. and Dannenhoffer, J. F., "Three-Dimensional Adaptive Grid Embedding Euler Technique," AIAA 93-0330, January 1993.

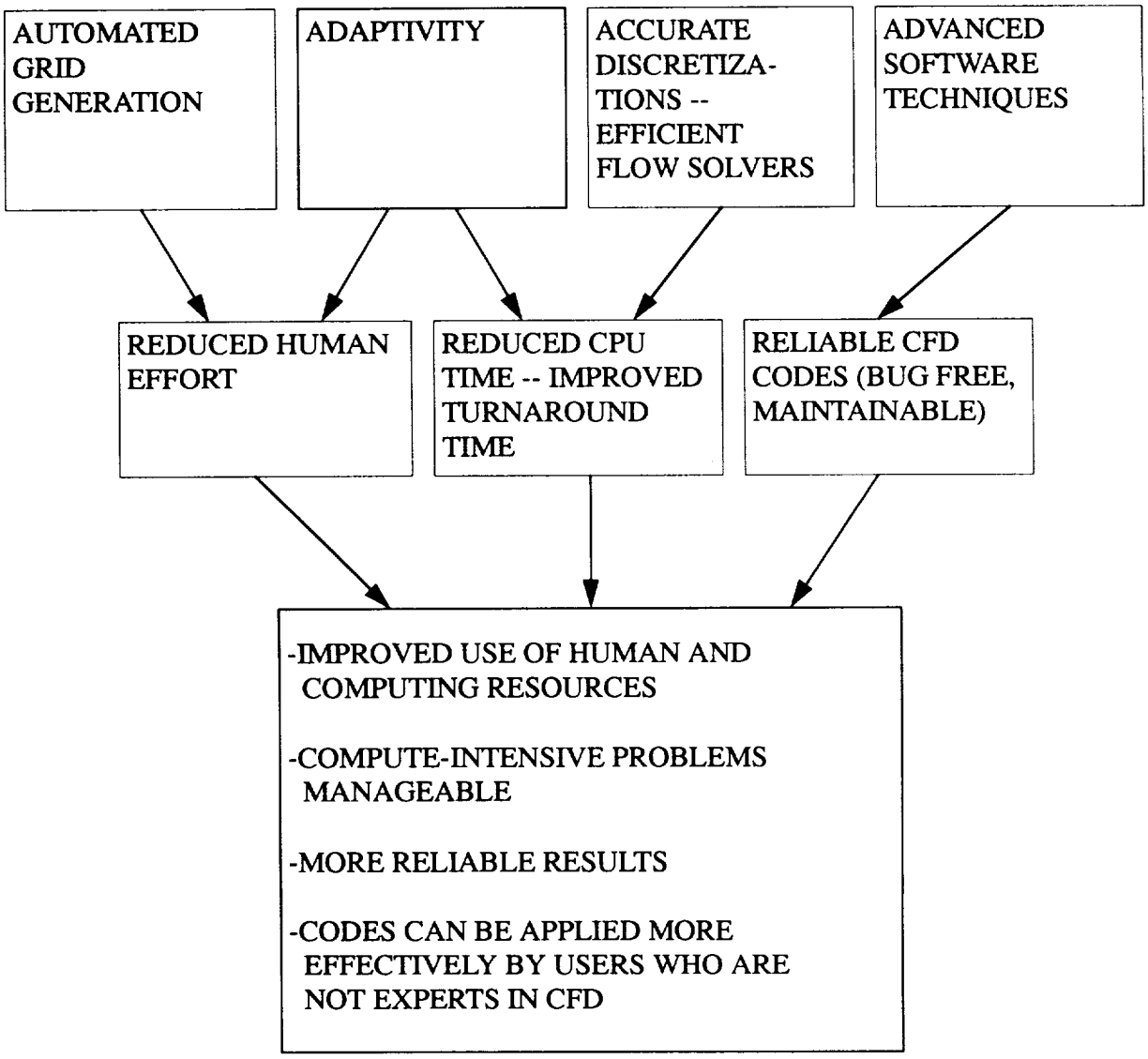
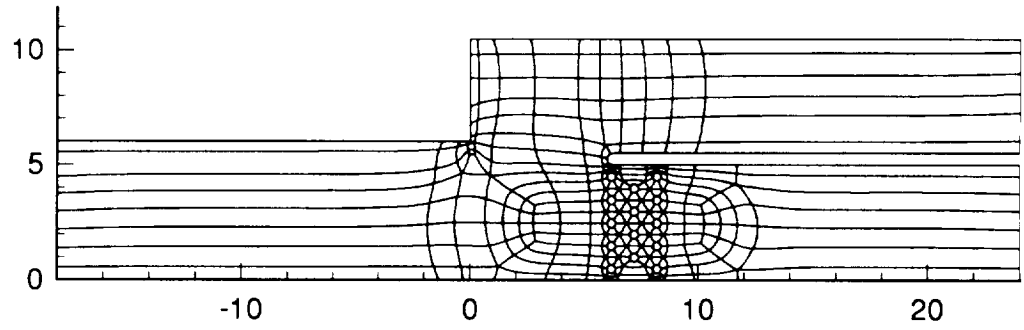
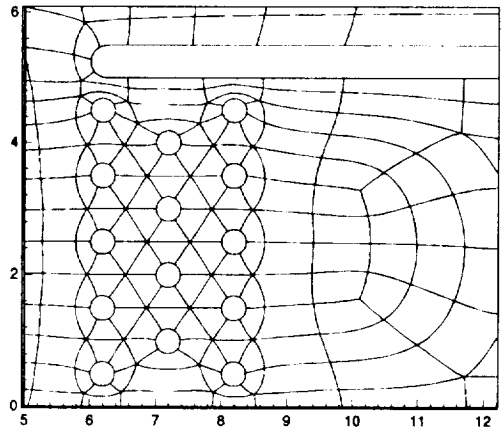



Figure 1 --- Ingredients and impact of an advanced methodology for simulation of viscous flows in complicated geometries




(a)




(b)

Figure 2. Topology of multiblock grid system (a) overall grid, (b) around pin array and partition

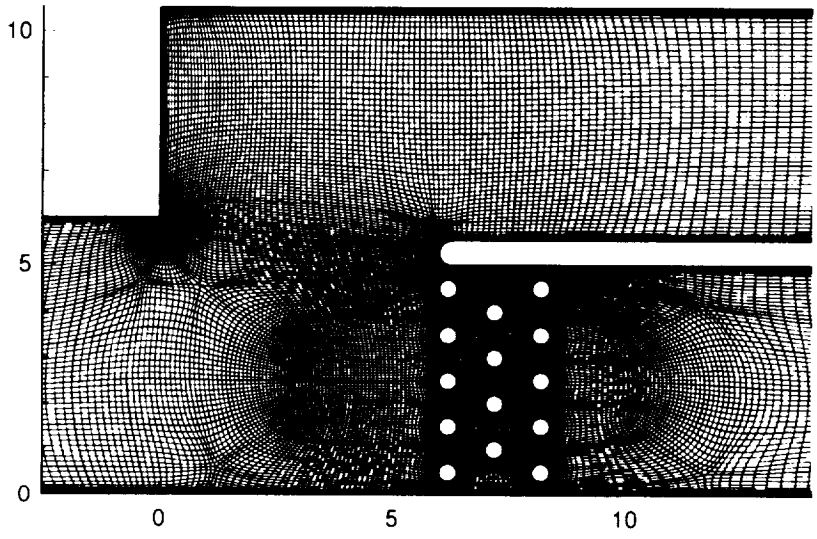
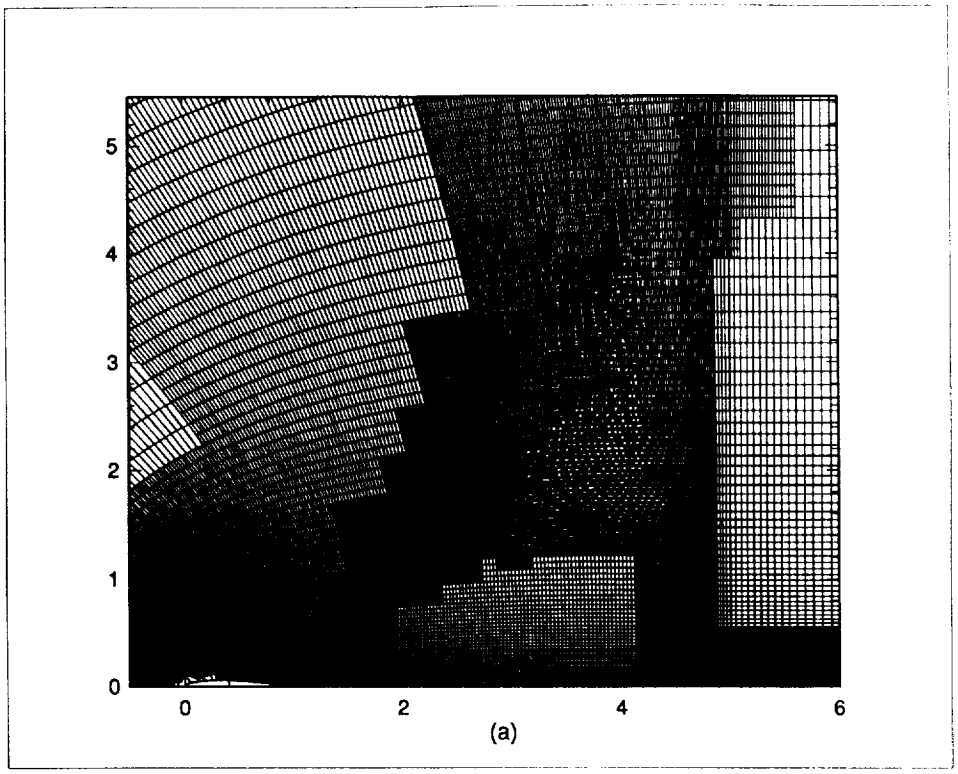



Figure 3. Grid system for the center region of the branched duct.



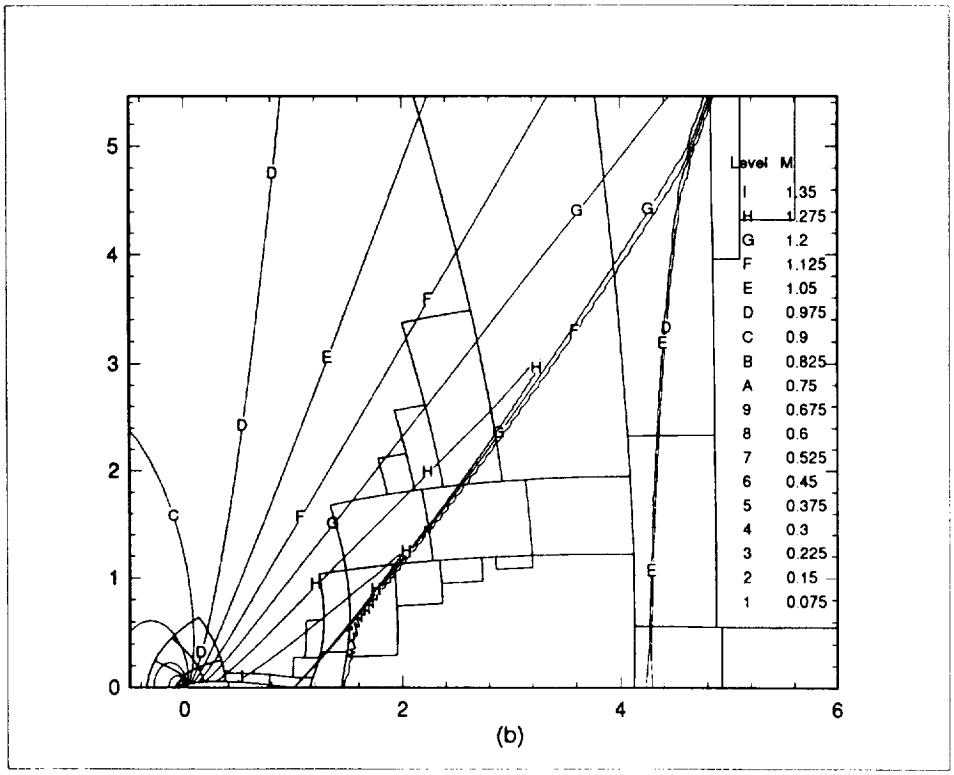




Figure 4. Transonic flow over a NACA0012 airfoil ($M = 0.95$); (a) adapted structured grid system, (b) contours of Mach number.

# NEW METHODS/APPROACHES/ APPLICATIONS (3)

N95-28767

# DEMONSTRATION OF AN AUTOMATED CFD SYSTEM FOR THREE-DIMENSIONAL FLOW SIMULATIONS[1]

J.W. van der Burg, J.E.J. Maseland, R. Hagmeijer, K.M.J. de Cock,
National Aerospace Laboratory NLR,
Anthony Fokkerweg 2,
1059 CM Amsterdam, The Netherlands.

## ABSTRACT

In this paper the capabilities of an automated CFD system which is currently available at NLR are demonstrated. Transonic flow around the AS28G wing/body configuration and hypersonic flow through a generic three-dimensional mixed-compression airbreathing inlet are simulated. An assessment of the level of automation of the current CFD-system is made. The problem-turnaround time lies within the order of a week for both applications.

## 1. INTRODUCTION

Experience with CFD technology (multiblock structured grid methods) learns that the turnaround time for generation of grids for Euler/Navier-Stokes calculations of complex aircraft configurations is large. In order to efficiently contribute to aircraft design the problem-turnaround time must be reduced to the order of a day or a week [1]. Unstructured grid methods offer the possibility to bring about this reduction.
The level of acceptance of CFD technology in the aerodynamic design process is directly related to the ability to produce accurate solutions. High accuracy of aerodynamic forces is especially important so that the computed lift, drag and pitching moment can be relied upon to reduce the risks involved in aircraft design.
In view of these aspects DLR and NLR started a cooperation entitled "CFD for complete aircraft" to develop a fully automatic system for three-dimensional flow simulations. The algorithms used are based on the unstructured grid approach [2] which is based on a Galerkin finite-element method to discretise the three-dimensional Euler equations.

The objective of this paper is to demonstrate the capabilities of the CFD system which is currently available at NLR. The focus will be on the level of automation and accuracy of the CFD system. The first application concerns three-dimensional transonic inviscid flow past the AS28G-wing/body configuration. The second application is the study of an airbreathing inlet, which is part of the AEOLUS programme, a joint industry project in the Netherlands [3]. The CFD system is used to simulate hypersonic inviscid flow (no real gas effects included) through a generic three-dimensional mixed-compression inlet with two ramps.

## 2. CFD SYSTEM AT NLR

At NLR an automated CFD-system for three-dimensional inviscid flow simulations is acquired. In the current CFD-system the following algorithmic steps are necessary to obtain a visual representation of the flow field for a given aircraft configuration.

1. Geometry definition.

[1]Part of this investigation has been carried out under contract awarded by the Netherlands Agency for Aerospace Programs



PRECEDING PAGE BLANK NOT FILMED

2. Surface triangulation.
3. Three-dimensional grid generation.
4. Pre-processing.
5. Flow calculation.
6. Grid adaption.
7. Visualisation and post-processing.

The algorithmic steps 2-6 are fully automated, i.e. extensive user interaction is not required for these steps. In this section these algorithmic steps are discussed.

## 2.1 Geometry definition

The geometry of an aircraft configuration is usually defined using a CAD system (e.g. ICEM-CFD) and described in a standard data format. Typically, the geometry is established in terms of geometric entities, viz. support surfaces and curves. The relation between these geometric entities, referred to as the topology, has to be defined explicitly. The respective entities are then modelled by means of piecewise cubic surface and curve representations.
At present an application starts either with a user-defined aircraft configuration or an aircraft configuration abstracted from a three-dimensional multi-block grid [4]. By adding the geometry description of far field boundaries and the symmetry plane (optional) a three-dimensional flow domain enclosed by bounding surfaces is defined. In such a way the boundaries of the flow domain are fully defined.

## 2.2 Surface triangulation

In order to perform a flow calculation the flow domain has to be discretised, i.e. boundaries are triangulated and a tetrahedral grid is generated in the interior of the flow domain. Firstly, the boundaries of the flow domain are discretised resulting in a surface triangulation. To this purpose a *surface triangulation algorithm* is employed which incorporates an equi-distribution algorithm for discretising curves and an advancing front-type generation algorithm for discretising surfaces. A distribution function controls the size of the edges and triangles in the surface triangulation.
This distribution function is defined by parameters which are specified in the background grid and in source terms. Source terms are introduced in regions to refine regions of special interest. The distribution function expresses a desired spacing in each point of the flow domain.

## 2.3 Three-dimensional grid generation

Subsequently, a three-dimensional grid of Delaunay-type is obtained by employing a *three-dimensional grid generation algorithm*. In this algorithm a prospective node is located at the barycenter of a tetrahedron. If this prospective node satisfies the Delaunay criterion the node is inserted and connected to the existing tetrahedral grid. In case the grid size, due to insertion of this node, becomes to small the prospective node is rejected. Smoothness of the three-dimensional grid is treated explicitly by the distribution function (as defined by the background grid and source terms). As a result a three-dimensional tetrahedral grid is generated which is bounded by a topologically two-dimensional surface triangulation.

## 2.4 Pre-processing

The next algorithmic step is the *pre-processing algorithm.* This algorithm is designed to optimise the flow calculation for the generated three-dimensional grid. By employing a colouring algorithm for the edges in the three-dimensional grid a high degree of vectorisation in the flow calculation can be reached.

### 2.5 Flow calculation

In the *flow calculation algorithm* three-dimensional inviscid flow is simulated. The incorporation of an upwind solver makes the CFD-system suited for the simulation of subsonic, transonic, supersonic as well as hypersonic flows. For the spatial discretisation of the three-dimensional Euler equations Roe's approximate Riemann solver [5] is utilised. Second-order accuracy is achieved by employing a MUSCL interpolation [6] for the state vectors, following the approach in [7, 8]. An entropy fix is incorporated to prevent physically incorrect features in a numerical solution. At present the boundary conditions are treated with first-order accuracy. Time integration is established by adopting a Runge-Kutta time step algorithm. Convergence acceleration is achieved by local time stepping and residual averaging.

### 2.6 Grid adaption

In case the flow calculation has sufficiently converged (by taking a large number of Runge-Kutta time steps) the *grid adaption algorithm* can be employed. The grid adaption algorithm is based on remeshing. Based on the values of an adaption variable (for instance: density, Mach number or pressure coefficient) an adaption indicator is calculated which is based on the undivided difference of this adaption variable. According to the value of the adaption indicator sources terms are generated which are utilised to adapt the surface triangulation. Subsequently, the three-dimensional grid generation algorithm is adopted to generate an entire new three-dimensional grid. By incorporating the newly generated source terms in the existing distribution function a locally refined grid is obtained. For the adapted grid a numerical solution is then obtained by first employing the pre-processing algorithm and subsequently the flow calculation algorithm. This process can be repeated until a numerical solution of sufficient accuracy is obtained.

### 2.7 Visualisation and post-processing

Finally, a *post-processing algorithm* is adopted to calculate aerodynamic quantities. Flow visualisation is achieved by using the commercial package Data Visualiser [9]. Interfaces are available to visualise the geometry definition, the surface triangulation and the numerical solution.

## 3. APPLICATIONS

This section discusses two applications of the current CFD system at NLR. The first application concerns transonic flow around a wing/body configuration and the second application concerns hypersonic flow for a generic three-dimensional mixed-compression airbreathing inlet configuration.

### 3.1 AS28G wing/body configuration

In the first application described here inviscid flow around the AS28G wing-body configuration is simulated. The geometry is defined by 14 support surfaces and 42 curves. The physical coordinates of these support surfaces and curves are abstracted from a multi-block grid. The support surfaces contain approximately 17000 nodes. Far field boundaries and a symmetry plane are incorporated to define a closed flow domain.
The surface triangulation shown in figure 2 is obtained. It can be observed that the nose region,

tail region, wing leading edge, trailing edge and the tip are refined. Source terms are located in these regions in order to obtain sufficient resolution of the geometry of the AS28G wing/body configuration and to reduce expected losses in total pressure.
One successive refinement is employed. The surface triangulation of the adapted grid is shown in figure 3. The dimensions of the surface triangulation and the three-dimensional grid for the initial and the adapted grid can be found in table 1. The problem size for the adapted grid is approximately two times larger in comparison with the initial. Cpu-times for the respective algorithmic steps are shown in table 2 and the convergence history is depicted in figure 1.
In the flow calculation a three-step Runge-Kutta time step scheme with a CFL-number of 0.9 is utilised. Two jacobi iterations are employed to smooth the residuals. The flow calculation algorithm uses 81 words per grid node, which is relatively large [10], and reaches a performance of 253 Mflops on the NEC SX3.
The Mach-number and total pressure distribution are shown in figures 4 and 5 respectively. A relatively small amount of grid points is necessary to resolve the lambda shock on the upper side of the wing. This is a promising result as for instance in a multi-block structured code more grid points are necessary to capture the same shock structure. Nevertheless, large losses in total pressure ($\pm 25\%$) are experienced at the leading edge of the wing. This is mainly due to fact that the boundary condition at the wing is only first-order accurate. A relative large loss in total pressure is also observed on the upper side of the wing (8%).

### 3.2 Three-dimensional mixed compression airbreathing inlet

One of the most critical enabling technologies of advanced reusable launchers is the propulsion system. For example a two-stage-to-orbit aerospace vehicle needs a propulsion system to power the vehicle from take-off to sustained flight at Mach numbers ranging from 6 to 7 (separation Mach number of the rocket powered second stage from the airbreathing first stage). Sustained flight of airbreathing aerospace vehicles at these Mach numbers is not feasible with todays technologies.
The inlet considered here is of mixed compression type [11, 12]. Before entering the inlet, the oncoming air is decelerated to a hypersonic Mach number lower than the flight Mach number by oblique shock waves (see figure 6). Near the narrowest duct cross section, the throat, the flow passes a normal shock wave. Behind this shock the flow is subsonic. Further deceleration is required to a velocity acceptable for the combustion chamber. The subsonic deceleration requires a diverging duct called the diffuser.
The present section discusses typical results for flow through the mixed compression airbreathing inlet. The geometry of the inlet is described in full detail in reference [13]. The inlet geometry is based on a two-dimensional design that has been obtained by using engineering tools. The inlet considered here is designed for a freestream Mach number of $M_\infty = 4.5$. The forebody of the space plane is not modelled. Hence, a Mach number smaller than the flight Mach number is considered. Bleed slots are not taken into account since viscosity is neglected. Both the cowl lip and the side wall are modelled with finite dimensions.
The geometry of the inlet is defined by 46 curves and 18 support surfaces. The geometry contains 16 planes and 2 cubic polynomial representations, namely the cowl lip and the leading edge of the side wall. Figure 7 shows an inside view of the inlet from the upstream direction. The cowl lip and the side walls are clearly visible. At the top of the figure two ramps can be observed. The large bounding planes at the right-hand and left-hand side are boundaries of the flow domain on which symmetry conditions are imposed. In the inlet duct the right-hand bounding plane is also defined as a symmetry plane. At the left-hand side the wall of the inlet duct can be observed. An impression of the surface triangulation on the side-wall (left-hand side of figure 7) is shown in figure 8. Sufficient grid points are generated to obtain an accurate representation of the cowl lip.

For the freestream Mach-number $M_\infty = 4.5$ a typical "unstart" condition is obtained, see figure 9. This means that there is a strong bow shock in front of the duct inflow plane providing subsonic flow throughout the duct. In windtunnel experiments a "started" condition is obtained by opening the throat significantly so that the shock can pass. Once the shock has been swallowed, the diffuser throat area is decreased, while simultaneously the pressure ratio of the pressure of the outlet of the diffuser with respect to the freestream pressure is increased so that the shock is caused to move to the diffuser throat [14].
During numerical simulations, however, the variation of the geometry is not trivial to simulate. To solve this problem the calculation is started with a higher Mach number $M_\infty = 5.0$ so that effective throat area is increased. This results in a globally correct pattern of oblique shock waves. The "unstart" condition is not experienced as the effective throat area is too wide for that particular Mach number. Subsequently, the Mach number is gradually lowered to the required value $M_\infty = 4.5$.
Moreover, to obtain the final shock in the diffuser throat and accordingly subsonic outflow, the pressure at the diffuser outlet has to be imposed. The location of the final shock is a function of the value of the pressure, which can be found by employing normal shock relations and using the fact that the flow is isentropic behind the final shock (see reference [15]).
Figure 10 shows a side view of the "started" condition with a final shock behind the throat of the duct. It can be observed that the oblique shocks and the final shock are resolved quite accurately.

## 4. LEVEL OF AUTOMATION

A reduction in CFD-problem-turnaround time can be achieved by an increased level of automation. This section will focus on the automation level of the current CFD system for the applications discussed in section 3. Those algorithmic steps in the current CFD system described in section 2 which require user-interaction are identified. This identification is necessary in order to assess the potential for an increased level of automation.

A With respect to geometry definition (step 1) an aircraft configuration is usually defined in terms of standard CAD data format. Extensive user interaction is required to generate a multi-block grid (such as for the wing/body configuration) which is suited as input for the current CFD-system. Consequently the problem-turnaround-time is large for multi-block based geometries.

B The definition of far field boundaries (step 1) (such as for the wing/body configuration) requires user interaction. The topology of a three-dimensional far field cube, the dimensions of the cube and the relation with between cube and the aircraft configuration have to be established.

C The definition of the distribution function (steps 2 and 3), which controls the size and shape of triangles in the surface triangulation and tetrahedral elements in the three-dimensional grid, requires user interaction. Visualisation of the generated surface triangulation is necessary in order to inspect the influence of the distribution function. A surface triangulation is accepted if sufficient resolution of details of the aircraft configuration (like wing leading and trailing edge) is obtained. After modification of the distribution function it is necessary to regenerate the surface triangulation. It is observed that the definition of a suitable distribution function does not guarantee a successful three-dimensional grid generation algorithm. An intolerably large number of tetrahedral elements may be generated in case parameters defining the distribution function are chosen too small. For the current CFD-system an expert user is required to specify the distribution function.

D It is necessary to define boundary types (like symmetry plane, far field or solid wall) for each surface of the flow domain (step 4). This requires only a relative small amount of user interaction.

E This requirement also holds for the specification of flow calculation parameters (step 5), such as for instance: Mach-number, angle of attack, CFL-number, number of Runge-Kutta time steps and Restart-option.

F Inspection of the flow solution by visualisation (step 7) can have a negative influence on the problem-turnaround-time if the number of nodes in the three-dimensional grid is large. This effect can be attributed to the fact that the handling of a large data-file in a multi-user environment and the operations necessary for visualisation, such as zooming, pivoting and moving, are relatively slow.

G In the grid adaption algorithm (step 6) user interaction is required to assess the level of grid refinement. Initially, default values for the grid adaption parameters are taken. Visual inspection then learns whether the surface triangulation has been adapted sufficiently. If the generated surface triangulation is not acceptable (for instance the level of refinement is too high) the grid adaption algorithm has to be repeated with modified input-parameters. Moreover, the number of nodes generated in the three-dimensional grid (after adaption, remeshing) is very sensitive to the values of the parameters specified in the grid adaption algorithm. If the number of nodes generated becomes too large it is necessary to repeat the grid adaption algorithm and the three-dimensional grid generation algorithm.

In the near future the focus is on the respective parts requiring extensive user interaction, namely items A, C and G. It is foreseen that the remaining items B, D and F requiring more than minimal user interaction can be automated.

By constructing an interface with a standard CAD data format the generation of a multi-block grid as input for the CFD-system can be avoided (item A). The data structure in the current CFD system is capable of handling a standard CAD data format. This would already significantly reduce the amount of user interaction.
The influence of the distribution function (background grid and source terms) and the grid adaption parameters on the three-dimensional grid generation algorithm remains difficult to estimate (item C and G). Ideally, a functional relationship between these parameters and the number of nodes generated in the three-dimensional grid could contribute here. However, for complex geometries this relationship is difficult to assess.

## 5. CONCLUSIONS

The capabilities of the CFD system have been demonstrated for two applications. For the AS28G wing/body configuration and the three-dimensional mixed compression airbreathing inlet configuration the results so far indicate the potential for short turnaround times. An assessment of those parts in the CFD system which require user interaction shows that the automation level can be increased. The actual problem-turnaround time (geometry definition, surface triangulation, 3D grid generation, flow calculation, grid adaption, flow visualisation and post-processing) lies in the order of a few days for both applications.
Moreover, the results indicate that the current CFD-system produces accurate solutions. The upwind-based flow calculation algorithm yields a spatially second-order accurate solution. However, the boundary conditions are currently only first-order accurate.

## References


[1] P.E. Rubbert, "CFD and the changing world of airplane design", In *ICAS Proceedings, 19th Congress of the International Council of the Aeronautical Sciences*, September, 1994.

[2] N.P. Weatherill, O. Hassan, D.L. Marcum, "Calculation of steady compressible flow fields with the finite element method", AIAA 93-0341, 1993.

[3] R. Sudmeijer, W.B. de Wolf, "Research, Design and Development plan for the inlet system of an airbreathing launcher", 1993.

[4] J.W. Boerstoel, "A system of CFD codes for industrial CFD analysis of flows around aircraft including propulsion systems modelling", NLR CR 93-519 L, 1993.

[5] P.L. Roe, "Approximate Riemann solvers, parameter vectors and difference schemes", Journal of Computational Physics, 43:357-372, 1981.

[6] B. van Leer, "Towards the ultimate conservative difference scheme. V. A second-order sequel to Godunov's method", Journal of Computational Physics, 32:101-136, 1979.

[7] J.W. van der Burg, "Numerical techniques for transonic flow calculations". PhD thesis, University of Twente, Enschede, The Netherlands, 1993.

[8] K.M.J. de Cock, "Multigrid convergence acceleration for the 2D Euler equations applied to high lift systems", NLR TP 93301, 1993.

[9] Wavefront Technologies, Inc., Data Visualizer User's Guide, Fourth Edition, May 1993.

[10] T.A. Kinard, B.W. Harris, "Evaluation of two unstructured CFD methods", AIAA 94-1877, 1994.

[11] R. Hagmeijer, "Required tools for aerothermodynamic analysis and design of space plane engine intakes", AE-NLR-TN-9303 (1), 1993.

[12] R. Hagmeijer, B. Oskam, J.W. van der Burg, J.E.J. Maseland, K.M.J. de Cock, "Aerothermodynamic models of hypersonic flow and their application to the inlet flow of an airbreathing launcher", 1994.

[13] R.H.J. Oerlemans, "Geometric description of a generic space plane intake for CFD calculations and preliminary structural design work", AE-NLR-TN-9403 (1) in progress, 1994.

[14] M.J. Zucrow, J.D. Hoffman, "Gas Dynamics", Volume 1. John Wiley and Sons, Inc., 1976.

[15] B.G. ten Wolde, "Flow calculations for the supersonic inlet of an airbreathing launcher", NLR-TR-94-375, 1994.

| | Surface triang. | | 3D Grid Generation | | | Flow calc. |
|---|---|---|---|---|---|---|
| Grid | nodes | triangles | nodes | elements | edges | time steps |
| Initial | 22275 | 44546 | 105786 | 609338 | 737396 | 500 |
| Adapted | 30744 | 61484 | 222359 | 1348274 | 1600926 | 2000 |

Table 1: Number of nodes and triangles in the surface triangulation, number of nodes and number of elements in three-dimensional grid for the initial grid and the adapted grid for the AS28G wing/body configuration

| Algorithm | cpu-time (initial grid) | cpu-time (adapted grid) | Mflops | computer |
|---|---|---|---|---|
| Surface triangulation | 31m 20s | - | 4 | SGI-Onyx |
| 3D grid generation | 5m 54s | 14m 16s | 4 | NEC SX3 |
| Pre-processing | 1m 18s | 3m 3s | 3 | NEC SX3 |
| Flow calculation | 46m 11s | 8h 5m 6s | 253 | NEC SX3 |
| Grid adaption | 2m 58s | - | 5 | NEC SX3 |

Table 2: Cpu-times for the respective algorithmic steps for the AS28G wing/body configuration

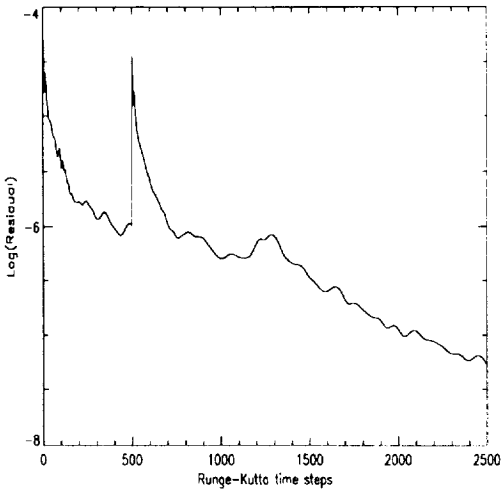



Figure 1: Convergence history of the flow calculation algorithm for the AS28G wing/body configuration on the initial grid and the adapted grid

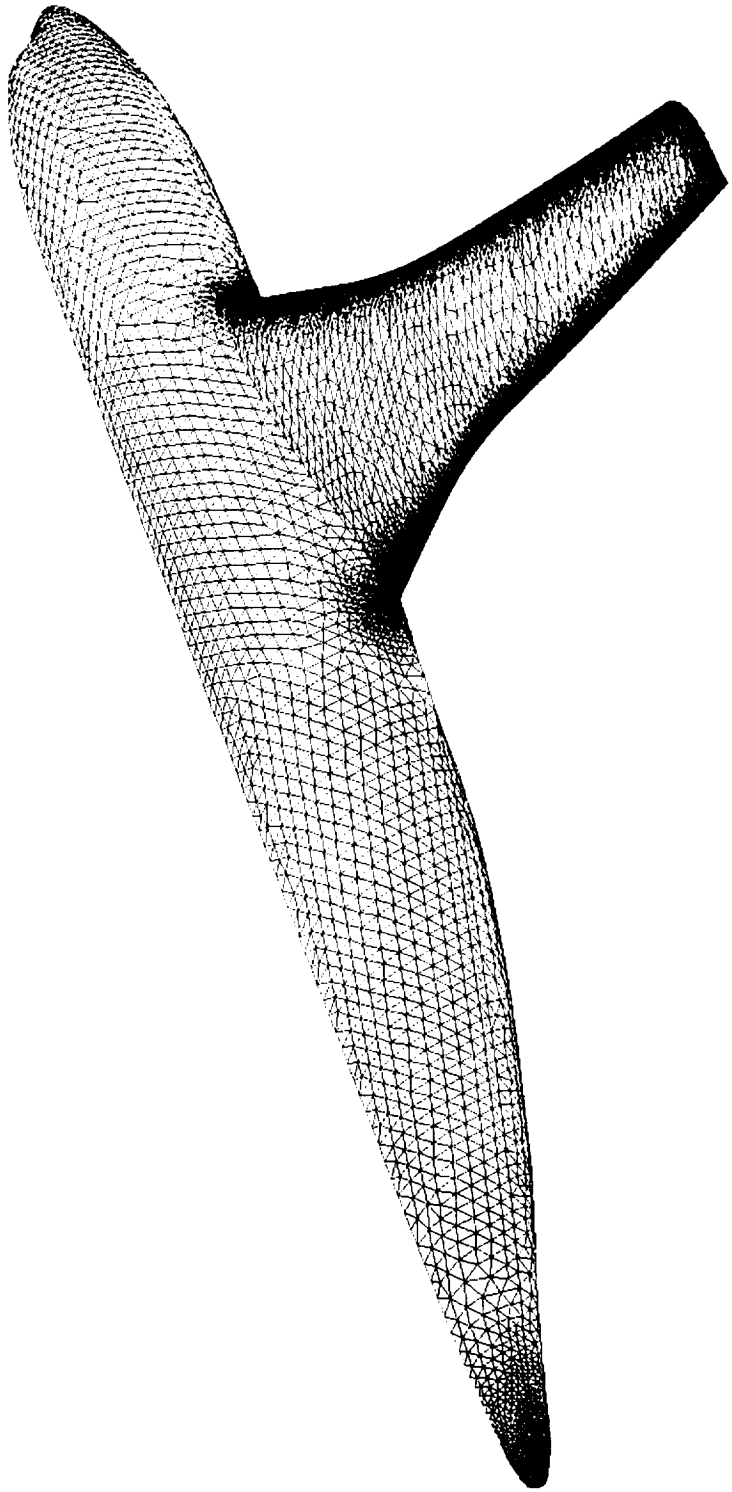


Figure 2: Surface triangulation for the AS28G wing/body configuration



ORIGINAL PAGE IS
OF POOR QUALITY

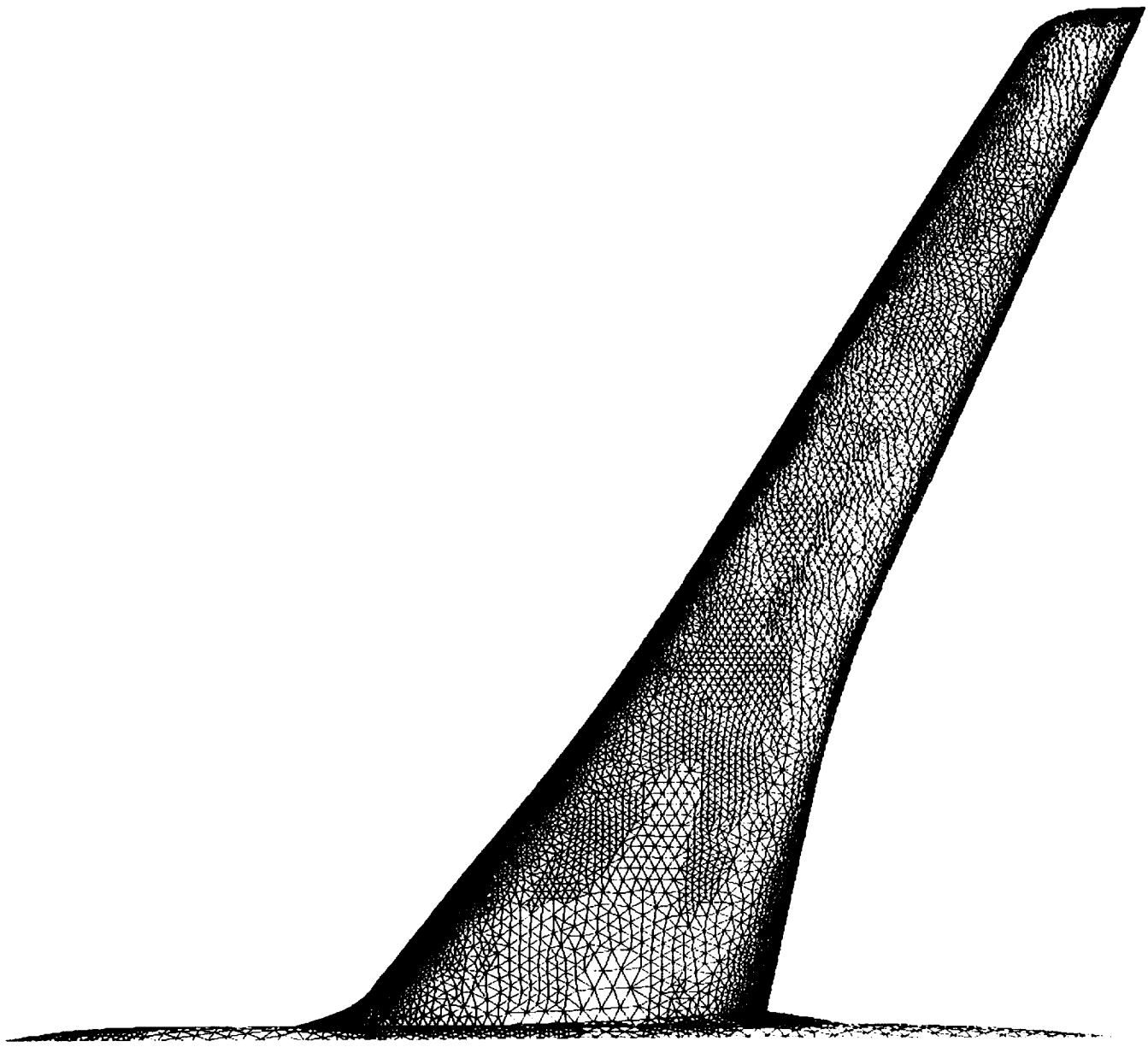


Figure 3: Surface triangulation for the AS28G wing/body configuration after one successive refinement



ORIGINAL PAGE IS
OF POOR QUALITY

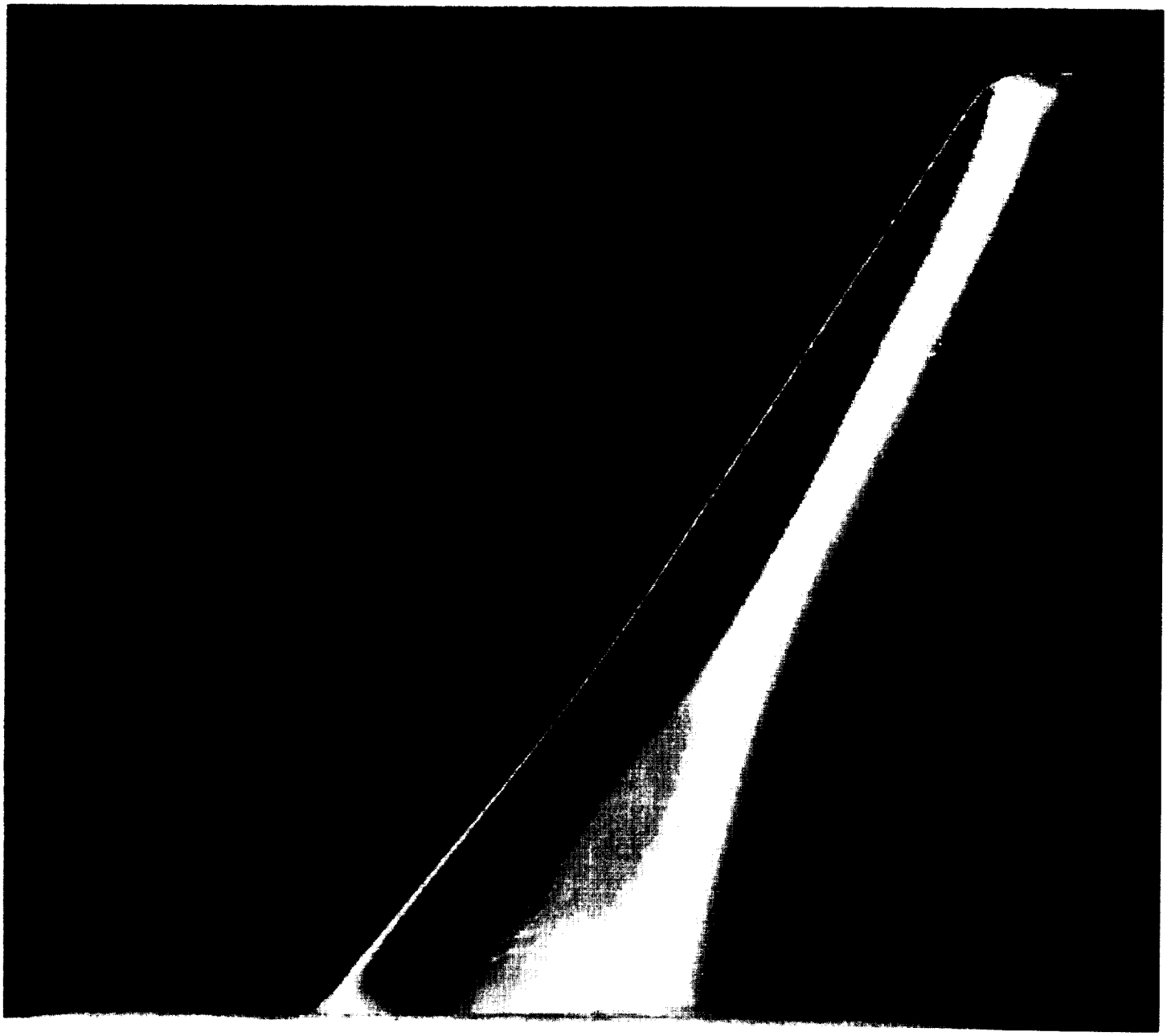


Figure 4: Mach number distribution for the AS28G wing/body configuration on the upper side of the wing after one successive refinement

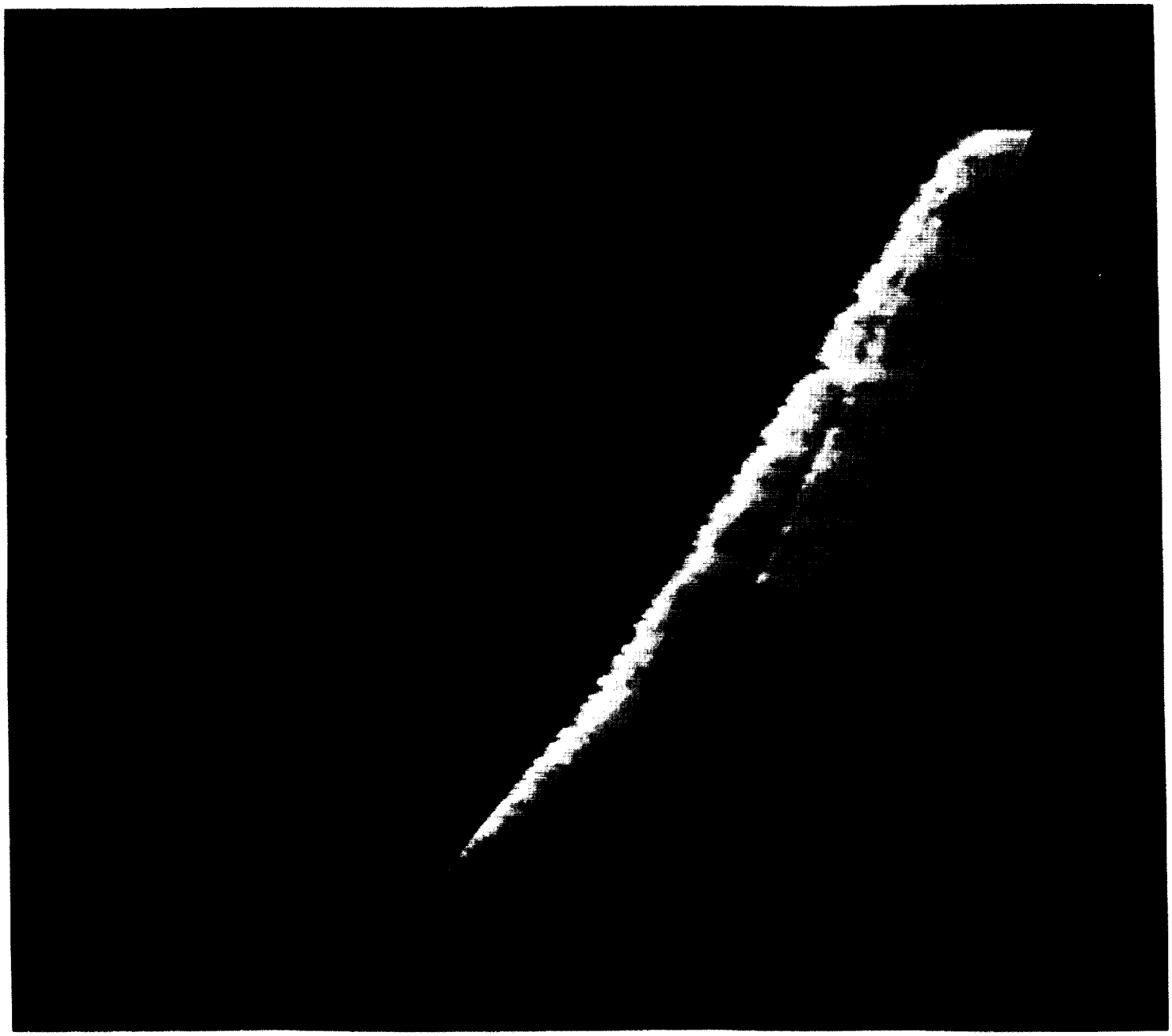


Figure 5: Total pressure distribution scaled with respect to the freestream values for the AS28G wing/body configuration on the upper side of the wing after one successive refinement

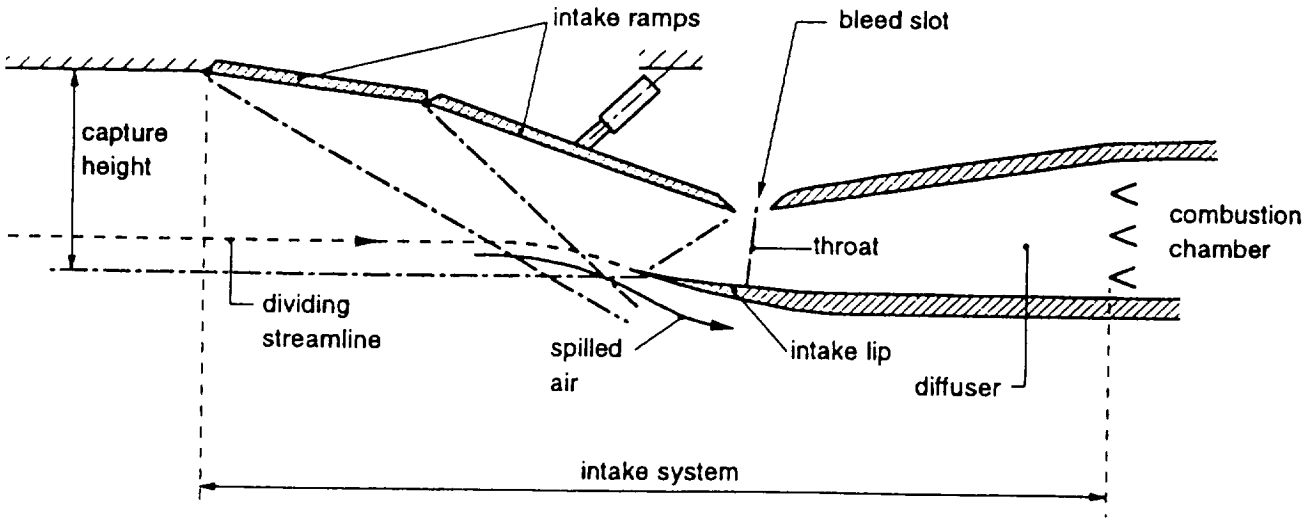



Figure 6: Main inlet components and terminology

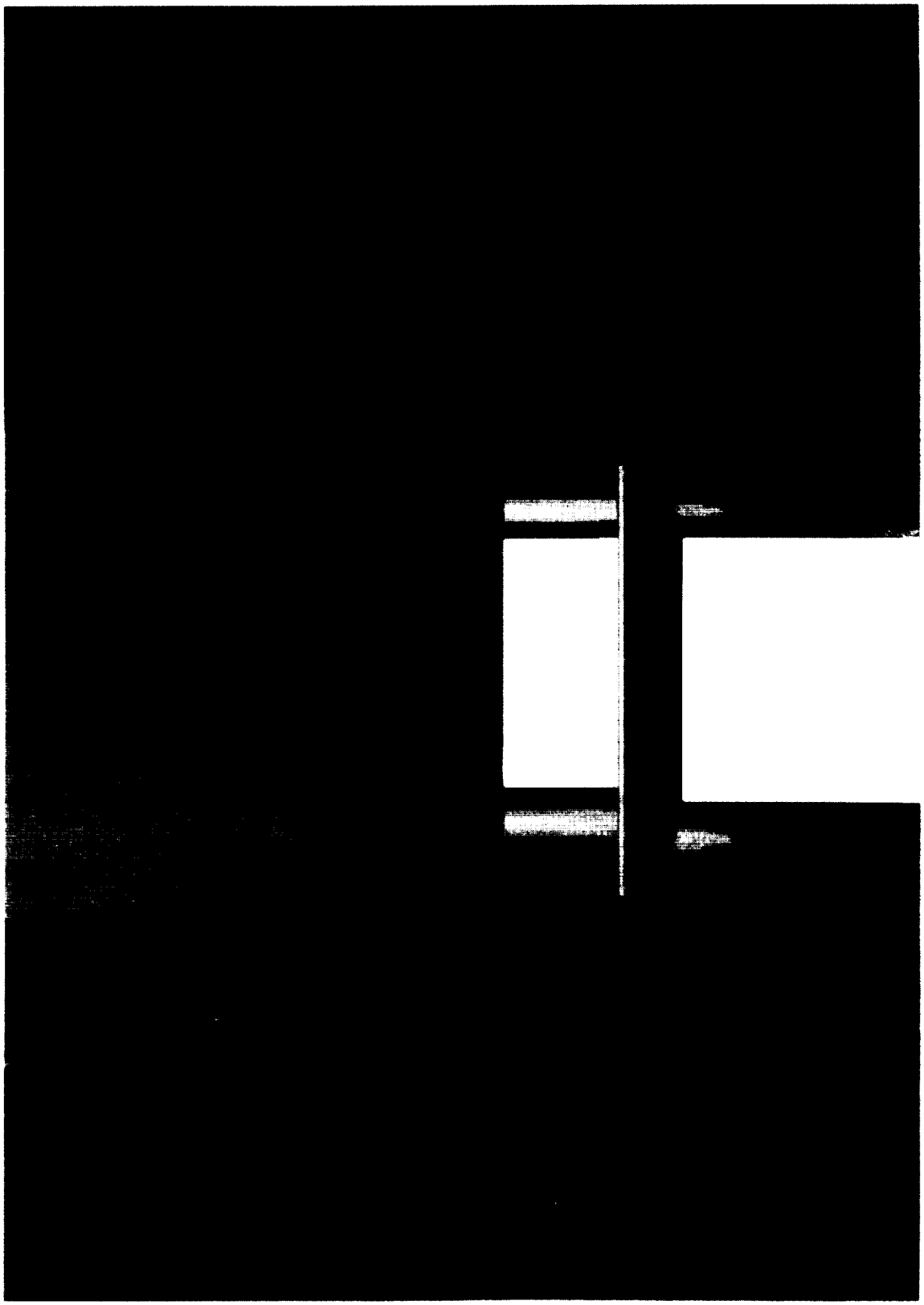


Figure 7: Impression of the three-dimensional mixed compression airbreathing inlet geometry modelled by cubic surface and curve representations

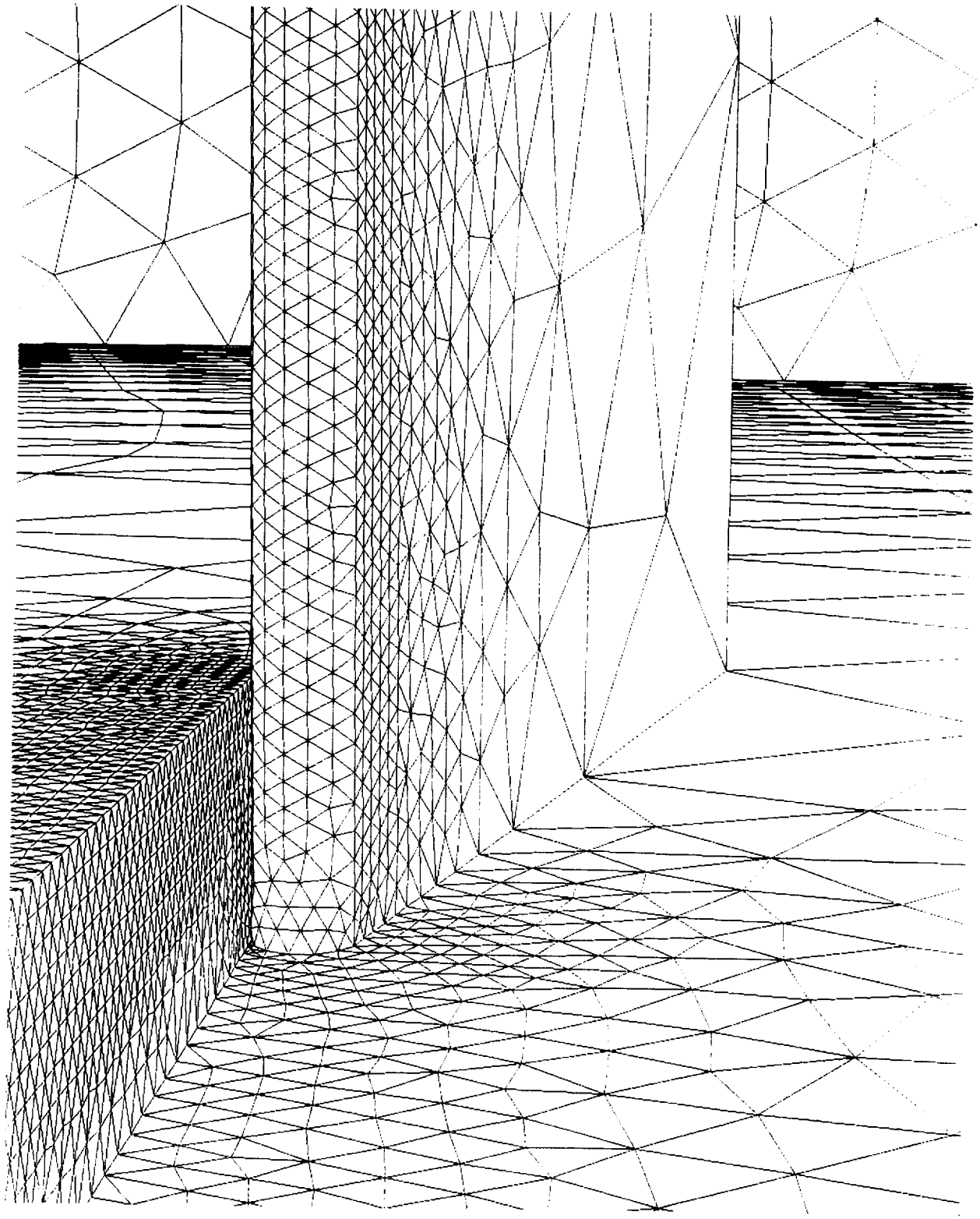


Figure 8: Surface grid at side-wall with refinements at the side wall and the cowl lip

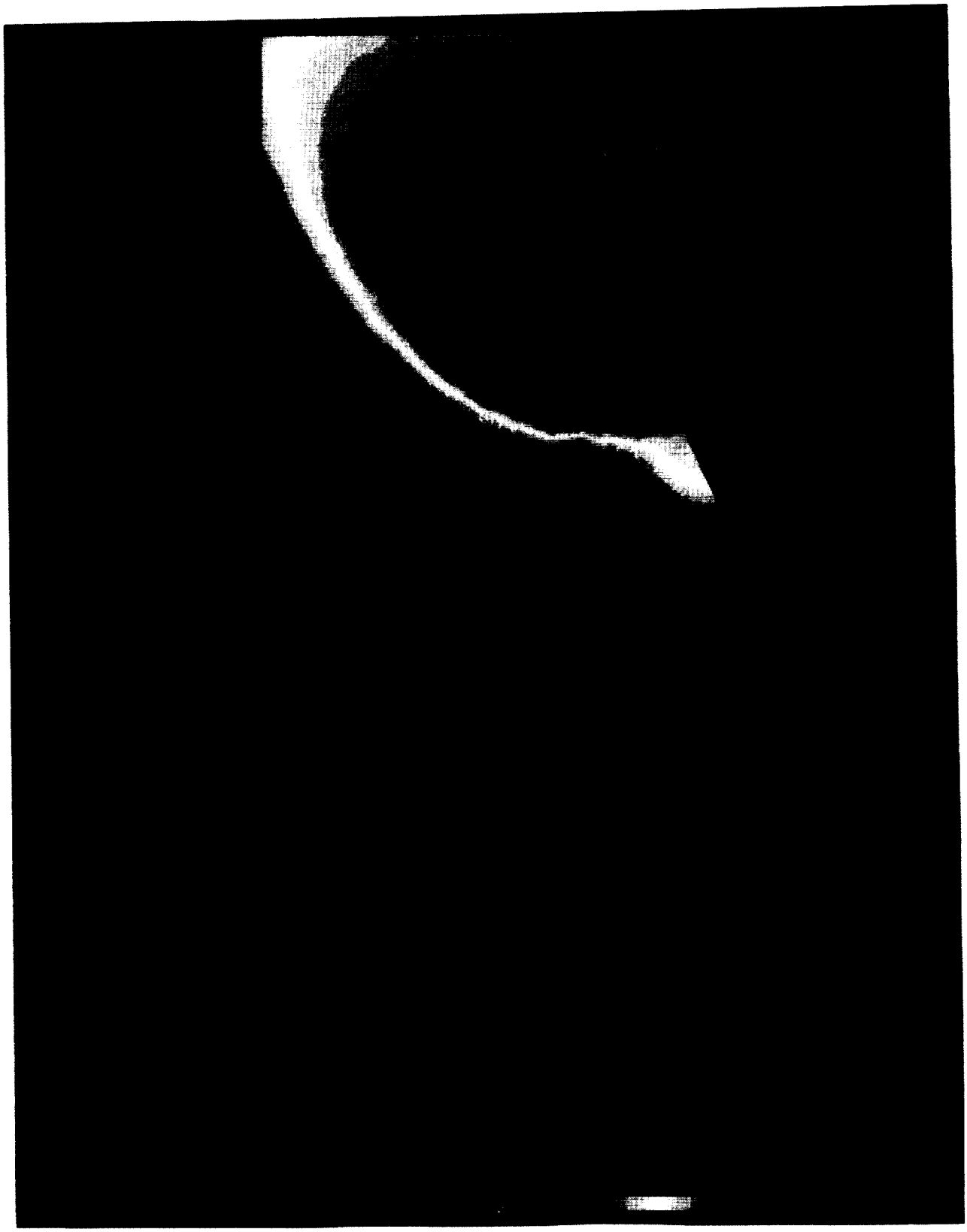


Figure 9: Side view of the Mach number distribution. The flow condition represents "unstarted" flow: strong bow shock and subsonic flow in duct

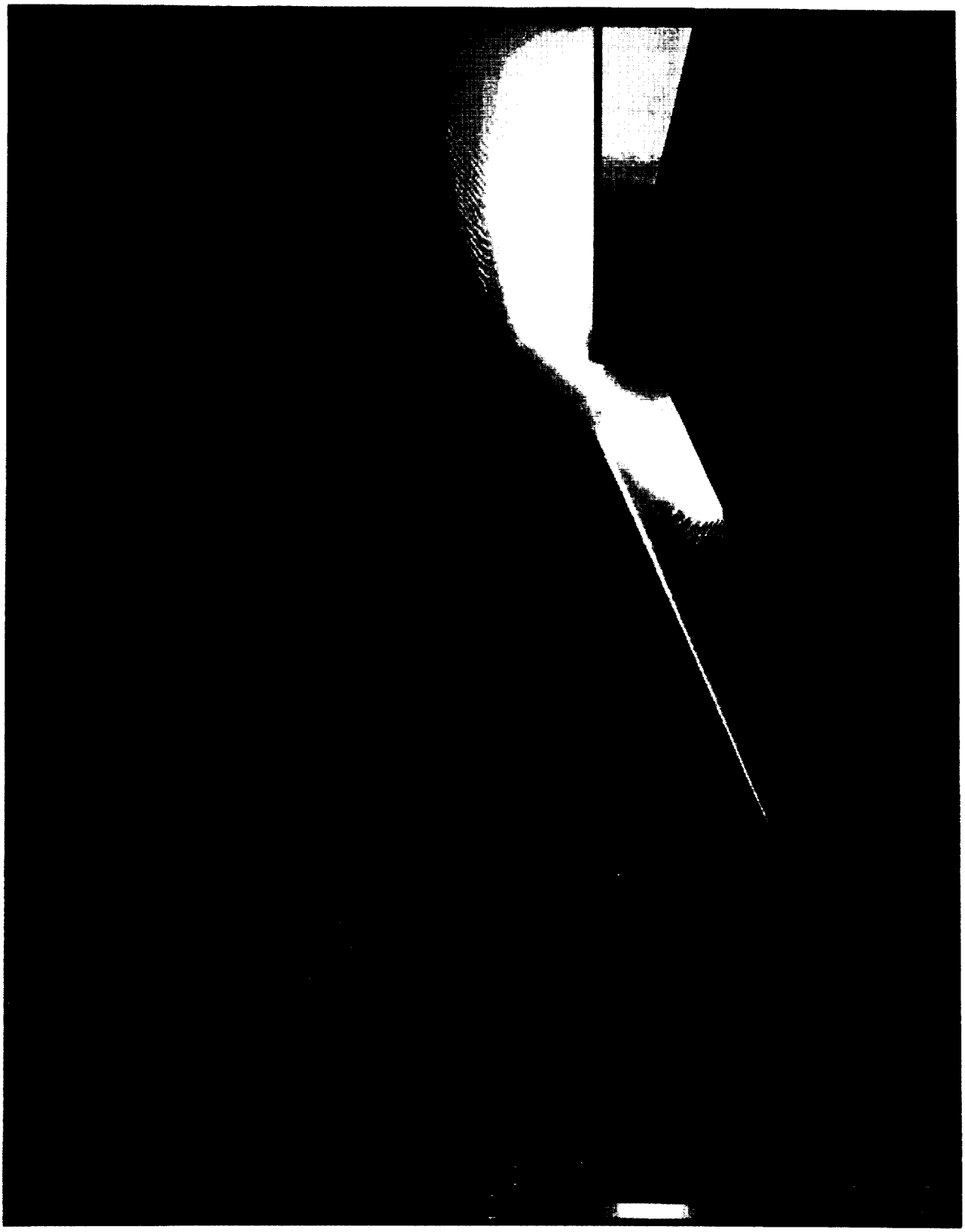


Figure 10: Side view of the Mach number distribution. The flow condition represents "started" flow with a final normal shock in the duct

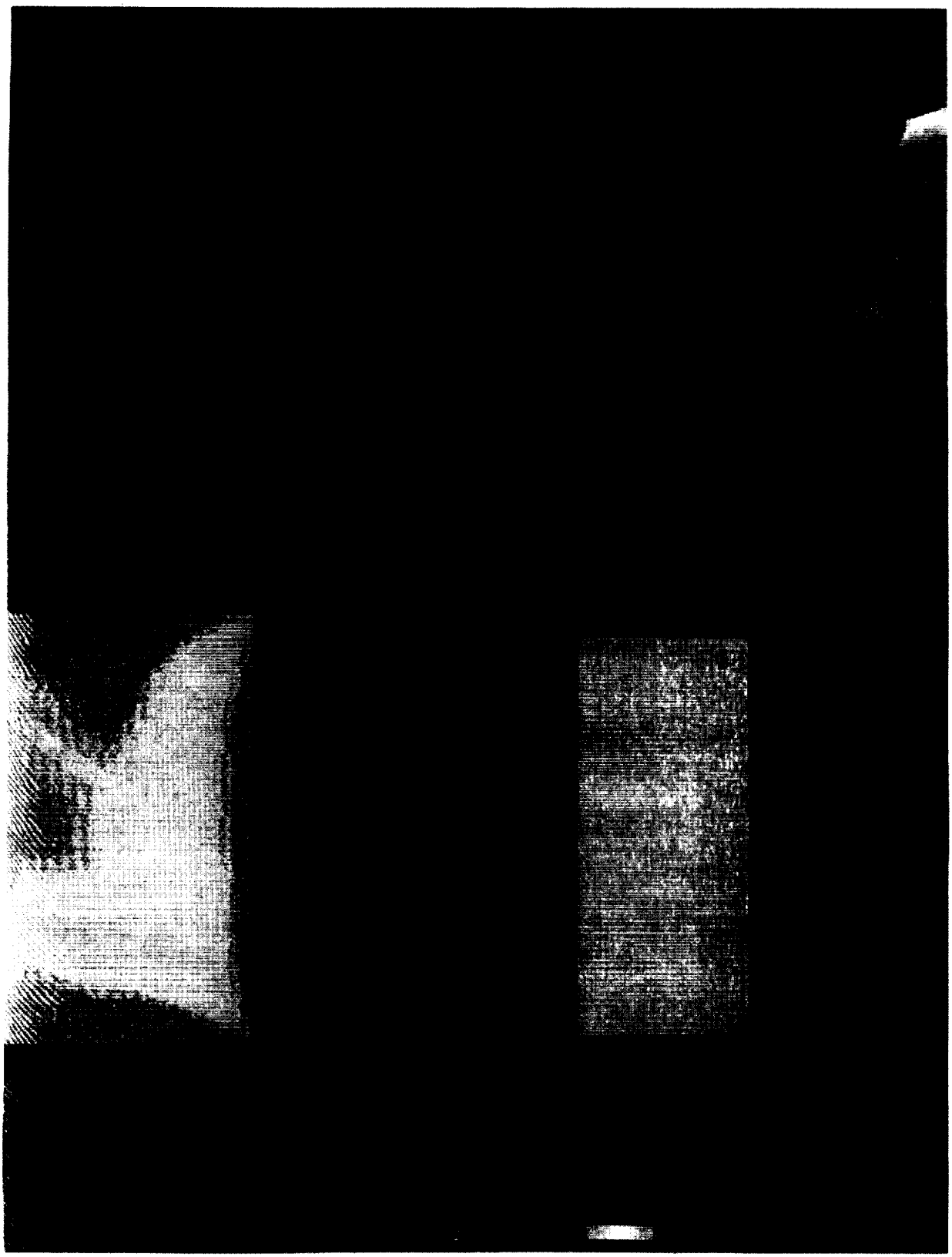


Figure 11: Detailed view of the Mach-number distribution at the intersection of the side-wall ramp and the cowl lip. The flow condition represents "started flow" with a final shock in the duct

N95- 28768

# A PROCEDURE FOR AUTOMATING CFD SIMULATIONS OF AN INLET-BLEED PROBLEM

Wei J. Chyu*
National Aeronautics and Space Administration
Ames Research Center
Moffett Field, California 94035

Mark J. Rimlinger¶ and Tom I-P. Shih§
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213

## SUMMARY

A procedure was developed to improve the turn-around time for computational fluid dynamics (CFD) simulations of an inlet-bleed problem involving oblique shock-wave/boundary-layer interactions on a flat plate with bleed into a plenum through one or more circular holes. This procedure is embodied in a preprocessor called AUTOMAT. With AUTOMAT, once data for the geometry and flow conditions have been specified (either interactively or via a namelist), it will automatically generate all input files needed to perform a three-dimensional Navier-Stokes simulation of the prescribed inlet-bleed problem by using the PEGASUS and OVERFLOW codes. The input files automatically generated by AUTOMAT include those for the grid system and those for the initial and boundary conditions. The grid systems automatically generated by AUTOMAT are multi-block structured grids of the overlapping type. Results obtained by using AUTOMAT are presented to illustrate its capability.

## INTRODUCTION

Effective control of shock-wave induced boundary-layer separation by using bleed is vital to the successful operation of inlets in supersonic aircraft. This is because the strong external/internal shock waves generated in a mixed compression inlet often induce boundary-layer separations that can cause the potentially dangerous unstart to occur. The importance of bleed in controlling shock-wave/boundary-layer interactions has led to a number of experimental and computational studies. Although all experimental studies showed that bleed can control shock-wave induced flow separation, they failed to show how bleed-hole geometry and flow conditions about the bleed holes influence the effectiveness of the bleed process. Computational studies of shock-wave/boundary-layer interactions with bleed fall into two groups. The first group modelled the bleed process by using boundary conditions which treated bleed holes collectively as a slot or as a porous surface without resolving the flow through the bleed holes (refs. 1 - 6). The second group studied the bleed process in detail by resolving the flow through each bleed hole (refs. 7 - 16). The second group, in early stages of their work, performed two-dimensional (2-D) studies in which each bleed hole is modelled as a slot on a flat plate (refs. 7 - 12). Later, three-dimensional (3-D) studies were performed where air in the boundary layer above a plate was bled through circular holes into a plenum (refs. 13 - 16). These studies showed the complex nature of shock-wave/boundary-layer interactions with bleed. These studies also showed that the bleed process is sensitive to many design and operating parameters with different ones dominating under different conditions.

Since a very large number of design and operating parameters need to be investigated for the inlet-bleed problem, computational fluid dynamics (CFD) simulation is an attractive approach to get the required information. But, in order for CFD to be useful and cost-effective, the turn-around time must be good in addition to obtaining meaningful results. There are several factors which affect the turn-around time of CFD simulations. Of these, grid generation is the most critical. For complicated geometries, the generation of good quality grids require not only substantial time, but also considerable expertise from the user. Thus, an automatic grid generator would greatly reduce turn-around time. To further reduce turn-around time, input

---

* Research Scientist, High-Speed Aerodynamics Branch, Applied Aerodynamics Division.
¶ Graduate Student, Department of Mechanical Engineering.
§ Professor, Department of Mechanical Engineering.

files that contain initial and boundary conditions as well as other information needed by the CFD code in order to generate solutions on the computer also should be generated automatically.

In this study, a CFD preprocessor, called AUTOMAT, was developed to reduce turn-around time for CFD simulations of an inlet-bleed problem. With AUTOMAT, once data for geometry and operating conditions have been specified (either interactively or via a namelist), it will automatically generate all input files needed to simulate the prescribed inlet-bleed problem by using the PEGASUS (refs. 17 and 18) and OVERFLOW (ref. 19) codes. The input files generated by AUTOMAT include those for the grid system as well as initial and boundary conditions. The inlet-bleed problem setup by AUTOMAT is one in which a number of geometric and operating parameters of importance to realistic inlet-bleed systems in supersonic aircraft can be investigated.

The organization of the rest of this paper is as follows. First, description is given on the inlet-bleed problem setup by AUTOMAT, including the geometric and operating parameters that can be varied for parametric studies. Next, the formulation and the numerical method of solution are outlined. This is then followed by a description of AUTOMAT, its operation and the built-in knowledge-based system. This paper concludes with some sample inlet-bleed problems setup by AUTOMAT along with some sample solutions for the flow field. A description of all input parameters into AUTOMAT is given in the Appendix.

## DESCRIPTION OF THE INLET-BLEED PROBLEM

The inlet-bleed problem of interest in this study involves a subsonic or supersonic turbulent boundary layer flowing past a flat plate, bleed of the boundary layer through one or more circular holes which can be normal or inclined and arranged in different patterns, and a plenum where the bleed is discharged. Figure 1 shows a typical example of such an inlet-bleed problem in which air above the flat plate is bled through four rows of inclined holes arranged in a staggered fashion. For this problem, the domain is the region bounded by the dashed lines which includes the region above the flat plate, the plenum, and a number of "half" bleed holes in the case of multiple rows of holes. Only "half" bleed holes needed to be included in the domain because of the symmetry in the spanwise direction.

The aforementioned inlet-bleed problem is selected for study because it enables the investigation of some of the most important design and operating parameters relevant to inlet-bleed systems in supersonic aircraft. More specifically, the following geometric parameters can be investigated (see Fig. 1):

- bleed-hole pattern (either one hole, three-holes in line, or rows of holes arranged in a staggered fashion),
- number of rows if staggered arrangement (N),
- spacing in streamwise direction ($L_x$, measured between centers of holes),
- spacing in spanwise Direction ($L_y$, measured between centers of holes),
- diameter of bleed holes (D),
- orientation or inclination of bleed holes ($\alpha$),
- thickness of flat plate ($L_t$), and
- plenum dimensions ($L_{P1}$, $L_{P2}$).

The operating parameters that can be investigated by this inlet-bleed problem include (see Fig. 1):

- with or without incident shock wave impinging on the boundary-layer next to flat plate,
- shock-wave-generator half angle ($\beta$)*,
- inviscid-shock-impingement point on flat plate (measured from the center of holes in the first row),
- freestream Mach number ($M_\infty$),
- freestream static pressure ($P_\infty$),
- freestream static temperature ($T_\infty$),
- adiabatic or isothermal wall,

* Note that the shock-generator half angle and the freestream Mach number must be such that the incident and reflected shock waves can be analyzed as a one-dimensional problem. Very weak oblique shocks become two-dimensional when approaching a flat plate.

- boundary-layer momentum thickness ($\theta$), and
- plenum back pressure ($P_b$).

## FORMULATION AND NUMERICAL METHOD OF SOLUTION

The inlet-bleed problem described in the previous section was modelled by the density-weighted, ensemble-averaged conservation equations of mass, momentum ("full compressible" Navier-Stokes), and total energy written in generalized coordinates and cast in strong conservation-law form. The effects of turbulence were modelled by the Baldwin-Lomax algebraic turbulence model (ref. 20).

In order to obtain solutions to the conservation equations, boundary and initial conditions are needed. The boundary conditions (BCs) setup by AUTOMAT for the different boundaries shown in Fig. 1 are as follows. At the inflow boundary where the flow is supersonic everywhere except for a very small region next to the flat plate, two types of BCs are imposed. Along segment A-B, all flow variables are specified at the freestream conditions except for the streamwise velocity (which has a turbulent boundary-layer next to the flat plate) and the stagnation temperature (which is kept constant everywhere including the boundary layer so that the static temperature varies in the boundary layer). The velocity profile in the turbulent boundary layer is described by using the method of Huang, et al. (ref. 21). Along segment B-C, post-shock conditions based on inviscid, oblique, shock-wave theory are specified. These post-shock conditions are also specified along the freestream boundary (segment C-D). At the outflow boundary where the reflected shock wave exit the computational domain, the flow is also mostly supersonic except for a small region next to the flat plate so that all flow variables are extrapolated. Here, linear extrapolation based on three-point, backward differencing was employed. When there are symmetry boundaries as shown in Fig. 1, the BCs imposed are zero derivatives of the dependent variables except for the velocity component normal to those boundaries which was set equal to zero. At the exit of the plenum where the flow is subsonic, a back pressure ($P_b$) is imposed, and density and velocity are extrapolated in the same manner as the variables at the outflow boundary. At all solid surfaces, the no-slip condition and adiabatic walls are imposed.

Even though only steady-state solutions are of interest, initial conditions are needed because the unsteady form of the conservation equations are used. The initial conditions employed in this study are as follows. In the region above the flat plate, the initial condition is the two-dimensional, steady-state solution for an incident and a reflected oblique shock wave on a flat plate based on inviscid, oblique, shock-wave theory. The streamwise velocity profile, however, is modified according to the method of Huang, et al. (ref. 21). This necessitated the density and static temperature to be modified as well in order to maintain constant stagnation temperature in the boundary layer. The initial conditions used in the bleed hole and plenum are stagnant air at constant stagnation temperature ($T_o$) and static pressure ($P_b$).

Solutions to the ensemble-averaged conservation equations along with the corresponding initial and boundary conditions are obtained by using the OVERFLOW code (ref. 19). The OVERFLOW code contains many algorithms. The one used in this study is as follows: Inviscid flux-vector terms in the $\xi$-direction are upwind differenced by using the flux-vector splitting procedure of Steger and Warming (ref. 22). Inviscid flux-vector terms in directions normal to the $\xi$-direction are centrally differenced in order to reduce artificial dissipation in those direction. Diffusion terms in all directions are also centrally differenced. The time-derivative terms are approximated by the Euler implicit formula since only steady-state or quasi-steady-state solutions are sought. The system of nonlinear equations that resulted from the aforementioned approximations to the space- and time-derivatives are analyzed by using the partially split method of Steger, et al. (ref. 23). In OVERFLOW, Jacobians and metric coefficients are interpreted as grid-cell volumes and grid-cell surface areas, respectively. In this regard, all algorithms in OVERFLOW are implemented in the finite-volume manner. However, BCs in OVERFLOW are implemented in a finite-difference manner in order to enhance flexibility and ease in investigating different problems. Grid systems that are needed to obtain solutions are described in the next section.

## THE PREPROCESSOR AUTOMAT

For the inlet-bleed problem shown in Fig. 1 and described in the previous two sections, once data describing the geometry and operating conditions have been specified, the preprocessor, AUTOMAT, will automatically generate all input files needed by PEGASUS and OVERFLOW to perform CFD simulations. The data input into AUTOMAT can be interactive or non-interactive. If it is interactive, then it relies on

PLOT3D (ref. 24) for the graphics output on the screen. Thus, PLOT3D must be available on whatever computer being used if interactive mode is desired. Whether the data input is interactive or non-interactive, outputs of AUTOMAT are as follows:

- a grid system in PLOT3D format for input to OVERFLOW (grid.in),
- an intial condition file in PLOT3D format for input to OVERFLOW (q.save),
- a data file for input to OVERFLOW (bleed.inp), and
- an input file for PEGASUS to create the fort.2 file needed by OVERFLOW for overlapping grids.

In the next section, how AUTOMAT interfaces with the user in an interactive session is described (see APPENDIX for a non-interactive session). Afterwards, the knowledged-based system built into AUTOMAT to generate grid systems is briefly outlined.

## User Interface

AUTOMAT is written in Fortan 77. Once installed on a computer, AUTOMAT is invoked by entering "automat" and then pressing return or enter. On computers such as an Iris workstation with PLOT3D available, the interface with the user can be interactive or non-interactive. If PLOT3D is not available, then only non-interactive sessions are possible. For interactive sessions, the user will be prompted on the screen for the input data. For non-interactive sessions, the inputs are all contained in a namelist file, which can be modified by the user via any editor (e.g., vi in unix). Here, it is noted that the namelist file required for non-interactive sessions is automatically generated after every interactive session to minimize effort needed to input data for future sessions.

A typical interactive session with AUTOMAT proceeds as follows:

>Enter "automat" to invoke program.

>interactive or non-interactive?
Enter y or yes for interactive, and n or no for non-interactive.
If interactive, then all inputs would be prompted for on the screen.
If non-interactive, then all inputs can be placed in a namelist file. After each interactive session, a namelist file is created automatically to facilitate future sessions.

>units (English or metric)?
Enter e for English, and m for metric.

>Requests for input on operating parameters one at a time (freestream Mach number, freestream static pressure, freestream static temperature, gas constant (universal gas constant / gas molecular weight), Reynolds number per unit length (based on freestream density, velocity, and viscosity), boundary-layer momentum thickness, with or without incident shock (y for shock, n for no shock), shock-generator half angle (only asked if there is an incident shock), plenum back pressure).

>Requests for verification on inputted operating parameters. Enter y or yes to accept inputs. Enter n or no to modify inputs.

>Requests for inputs on geometric parameters one at a time (bleed-hole pattern (either one hole or rows of holes arranged in staggered fashion), bleed-hole diameter, bleed-hole inclination, number of rows, spacing between hole centers in streamwise direction, spacing between hole centers in spanwise direction, plate thickness, shock impingement location on flat plate under inviscid condition (measured from the center of holes in the first row), plenum dimensions).

>Requests for verification on inputted geometric parameters. Enter y or yes to accept inputs. Enter n or no to modify inputs.

>Shows grid generated in bleed hole on workstation.

>Requests for acceptance. Enter y or yes to accept grid generated. Enter n or no to modify grid generated.

>Shows grid generated above flat plate on workstation.

>Requests for acceptance. Enter y or yes to accept grid generated. Enter n or no to modify grid generated.

>Shows grid generated in plenum on workstation.

>Requests for acceptance. Enter y or yes to accept grid generated. Enter n or no to modify grid generated.

>Requests for amount of overlap above and below flat plate. Enter number of grid lines.

>Program outputs grid.in, q.save, bleed.inp, and an input file for PEGASUS, and then terminates.

This completes a typical interactive session with AUTOMAT. As noted earlier, sessions can be interactive or non-interactive. Non-interactive sessions do not show plots of the grid system generated on the screen, but provide the same outputs as the interactive sessions (namely, grid.in, q.save, and bleed.inp needed by OVERFLOW, and an input file for PEGASUS to get the fort.2 file which is also needed by OVERFLOW). Typically, an interactive session on a workstation requires about five minutes. Non-interactive session require less time (in fact, only only a few seconds on the Cray).

At the conclusion of a successful session with AUTOMAT, a job can be submitted to the Cray computer to run PEGASUS and then OVERFLOW. A typical grid system generated by using AUTOMAT for the inlet-bleed problem shown in Fig. 1 is given in Figs. 2 and 3. Typical solutions obtained by using OVERFLOW and PEGASUS are given in Figs. 4 and 5.

## Internal Knowledge-Based System

This section outlines how AUTOMAT uses the input information to generate the grid system needed to obtain solutions.

The domain size is based on the following inputs: bleed-hole pattern (single hole or rows of holes arranged in staggered fashion), bleed-hole diameter, bleed-hole inclination, number of rows, spacing between hole centers in streamwise direction, spacing between hole centers in spanwise direction, plate thickness, with or without incident shock, shock-wave-generator half angle, and inviscid-shock-impingement point on flat plate.

Some rules in computing domain size (namely, L and H in Fig. 1) are as follows:

- AB $\geq$ 4 (boundary-layer thickness),
- AB $\approx$ (2/3) AC,
- EF $\approx$ (1/3 to 2/3) DF,
- the leading and trailing holes are at least five diameters away from the inflow and outflow boundaries, respectively.

Before describing the rules used to determine grid spacings, it is emphasized that grid generation has the greatest influence on the turn-around time of CFD simulations. Thus, it must be automated as much as possible in order to reduce turn-around time. The ability to automate grid generation depends very much on the grid structure selected. For the inlet-bleed problem, a grid structure based on overlapping, multi-block structured grids has a number of attractive features that make complete automation of the grid generation process possible. This is because if grids can overlap, then the domain regardless of its geometric complexity can be partitioned and molded into sub-domains with simple geometries that make both grid generation and hence its automation easy. Also, because each grid of the overlapping grid is optimized for a particular region, each of those grids is of high quality in terms of resolution, smoothness, and orthogonality. However, overlapping grids are not without problems. Many users have reported difficulties with conservation errors because the interpolation schemes used to transfer data between overlapping grids are not conservative. Others have observed errors associated with increased artificial diffusion in the overlapped region due to sudden changes in grid alignment and spacing. Thus, care must be exercised while generating such grids in order to minimize these errors.

In AUTOMAT, the grid structure used is the overlapping grid (see Figs. 2 and 3). Basically, an H-H grid structure is used for the region above the flat plate and in the plenum. In the bleed hole, a combination of O-H and H-H structures are used, and they overlap each other. The grids in the bleed holes also overlap the H-H grids above the flat plate and in the plenum. The reason for using the H-H structure above the plate and in the plenum is obvious. For the bleed hole, the reason for using the O-H structure is to capture the geometry of the hole boundary, and the reason for using the H-H structure at the hole center is to eliminate the centerline singularity associated with O-H grids.

The grid spacings in AUTOMAT are based on the following inputs: bleed-hole pattern (single hole or rows of holes arranged in staggered fashion), bleed-hole diameter, bleed-hole inclination, number of rows, spacing between hole centers in streamwise direction, spacing between hole centers in spanwise direction, with or without incident shock, shock-wave-generator half angle, and inviscid-shock-impingement point on flat

plate, boundary-layer momentum thickness, $y^+$ of first few grid point above flat plate, amount of cluster about incident and reflected shock, and amount of protrusion of grid in bleed hole above and below plate.

Some rules used to determine grid spacings are as follows (see Figs. 1 to 3):

- If results are to be compared for several different bleed-hole configurations, then the grid above the plate must be the same for all bleed-hole configuration. This is necessary in order to ensure that the flow above the plate is the same in the absence of bleed. Thus, the case with the largest bleed zone (i.e., from leading edge of first hole to trailing edge of last hole) will be used to generate grid above the flat plate.

- Above the flat plate, the grid system has an H-H structure.

- In a region about the bleed holes (i.e., 2D before and after the first and last hole), the grid spacing in the streamwise direction is uniform, at least for all practical purposes because of the following constraint: grid lines pass through the leading edge, center, and trailing edge of each bleed hole along the two symmetry boundaries. This constraint was imposed in order to facilitate plotting of the surface pressure along a line in the streamwise direction that passes through the centers of the bleed holes.

- Away from the bleed zone, the grid spacings in the streamwise direction first become coarser and coaser, and then nearly uniform as the inflow and outflow boundaries are approached.

- In the direction normal to the flat plate, grid points are clustered next to the wall to resolve the boundary layer. The first five grid points away from the wall are set so that $y^+$ is equal to 1, 2, 3, 4, and 5.

- Grid points are also clustered about the incident and the reflected shock waves in order to maintain their "sharpness" before and after interacting with the boundary layer.

- The grid in the plenum is also an H-H grid with grid points clustered near all walls and the bleed holes.

- The grid in the bleed hole is made up of two overlapping grids -- an O-H grid to capture the geometry of the hole boundary, and an H-H grid at its center to eliminate the centerline singularity associated with O-H grids.

- Grid spacings for the bleed-hole grids in directions normal to the plate match those above and below the flat plate in the region where they protrude above and below the plate.

- Grid spacings for the O-H grid in the bleed hole are clustered next the wall of the bleed hole.

- The aspect ratio of the grids is near unity for the grid in the bleed hole, especially in the region near the top of the flat plate.

- When $\alpha$ is not 90 degrees, grid spacing in the azimuthal direction is based on identical arc length, not degrees.

Additional details about the clustering as well as other aspects of the automated grid generation process can be obtained by perusing the Fortan program.

## RESULTS

To illustrate the capabilities of AUTOMAT, the following steps were taken. First, AUTOMAT was used to generate several grid systems to demonstrate that geometric parameters such as bleed-hole inclination and the number of bleed holes can indeed be varied in parametric studies of the inlet-bleed problem. Next,

solutions were obtained to show that AUTOMAT does indeed provide all inputs files needed to do CFD simulations of the inlet-bleed problem and that the grid system generated can give meaningful solutions.

Figures 2 and 3 show the multi-block grid system generated by AUTOMAT for an inlet-bleed problem with four rows of normal bleed holes arranged in a staggered fashion. To further illustrate the automatic grid generation capabilities of AUTOMAT, grid systems were generated for the following two inlet-bleed problems without an incident shock wave impinging on the boundary layer: eight rows of normal bleed holes (Fig. 6), and eight rows of slanted bleed holes (Fig. 7). From these two figures, it can be seen that similar to the grid system shown in Figs. 2 and 3, grid lines are boundary conforming and are clustered near all solid walls whether the bleed holes are normal or slanted. Also, there are a sufficient number of grid lines clustered ahead of the first row of bleed holes and after the last row of bleed holes.

Figures 4 and 5 show the solutions obtained from PEGASUS and OVERFLOW by using the grid system (shown in Figs. 2 and 3) and other input files generated by AUTOMAT for an inlet-bleed problem with four rows of normal bleed holes arranged in a staggered fashion. These solutions have been shown to be grid independent as far as surface pressure and bleed rates into the bleed holes are concerned (ref. 16).

To further illustrate the parameters -- both geometric and operating -- that can be investigated, solutions were obtained for an inlet-bleed problem in which the bleed holes can be normal or slanted and the number of bleed holes can be one or three in tandem for a range of pressure ratios ($P_b/P_\infty$). For the cases with a single bleed hole, the incident oblique shock wave was adjusted so that it would impinge at the center of the bleed hole under inviscid conditions. For the cases with three-holes-in-tandem, that shock wave was adjusted to impinge at the center of the middle hole. The grid systems used in this parametric study are not shown, but can be found in ref. 15. Some of the results obtained for this parametric study are shown, and are given in Figs. 8 to 13. Figures 8 and 9 show the effects of bleed-hole inclination on the surface pressure as a function of $P_b/P_\infty$. Figures 10 and 11 show the effects of bleed-hole inclination on the surface pressure when there are three holes in tandem. Figures 8 to 11 can be compared with each other to examine the effects of having an upstream and a downstream bleed hole on the bleed process. Figures 12 and 13 show the flow coefficient in each bleed hole as a function of $P_b/P_s$, where $P_s$ is the average static pressure over the bleed hole when there is no bleed ($P_s$ has a different value over a different bleed hole due to the incident and the reflected shock waves). The flow coefficient is defined as the actual bleed rate divided by an ideal bleed rate. The ideal bleed rate assumes sonic flow in the entire bleed hole with stagnation pressure and temperature equal to those outside of the boundary layer. Analysis of the results shown in Figs. 8 to 13 can be found in ref. 15, and hence will not be repeated here.

The results presented in this section demonstrate that AUTOMAT can be used to automate CFD simulations of an inlet-bleed problem. These results also show that a relatively wide range of geometric and operating parameters of importance to inlet-bleed systems in supersonic aircraft can be investigated. The fast turn-around time afforded by AUTOMAT through the automatic generation of grid systems and other input files enables a broader range of design variables to be investigated.

## SUMMARY

A CFD preprocessor, called AUTOMAT, was develop to generate all input files needed by OVERFLOW and PEGASUS to generate solutions for an inlet-bleed problem that possesses many of the most important geometric and operating parameters relevant to realistic inlet-bleed systems in supersonic aircraft. This paper described the inlet-bleed problem that AUTOMAT automatically sets up by along with the geometric and operating parameters that can be investigated. This paper also described the user interface as well as the knowledge-based system built into AUTOMAT to generate grid systems. Results obtained by using AUTOMAT, namely grid systems and computed solutions for the flowfield, demonstrated the usefulness of AUTOMAT.

## ACKNOWLEDGEMENT

The research was supported through the NASA - Ames University Consortium Program. The authors are grateful for this support.

## REFERENCES


1. Abrahamson, K.W. and Bower, D.L., "An Empirical Boundary Condition for Numerical Simulation of Porous Plate Bleed Flows," AIAA Paper 88-0306, January 1988.

2. Benhachmi, D. Greber, I., and Hingst, W., "Experimental and Numerical Investigation of an Oblique Shock Wave/Boundary Layer Interaction with Continuous Suction," AIAA Paper 89-0357, January 1989.

3. Chyu, W.J., Howe, G.W., and Shih, T.I-P., "Bleed Boundary Conditions for Numerically Simulated Mixed-Compression Supersonic Inlet Flows," AIAA Journal of Propulsion and Power, Vol. 8, 1992, pp. 862-868.

4. Paynter, G.C., Treiber, D.A., and Kneeling, W.D., "Modelling Supersonic Inlet Boundary Layer Bleed Roughness," AIAA Journal of Propulsion and Power, Vol. 9, No. 4, 1993, pp. 622-627.

5. Lee, J., Sloan, M.L., and Paynter, G.C., "Lag Model for Turbulent Boundary Layers over Rough Bleed Surfaces," AIAA Journal of Propulsion and Power, Vol. 10, No. 4, 1994, pp. 562-568.

6. Harloff, G.J. and Smith, G.E., "On Supersonic-Inlet Boundaru-Layer Bleed Flow," AIAA Paper 95-0038, January 1995.

7. Hamed A. and Lehnig, T., "An Investigation of Oblique Shock/Boundary Layer/Bleed Interaction," AIAA Paper 90-1928, July 1990.

8. Hamed, A. and Lehnig, T., "The Effect of Bleed Configuration on Shock/Boundary Layer Interactions," AIAA Paper 91-2014, June 1991.

9. Hamed, A., Shih, S.H., and Yeuan, J.J., "An Investigation of Shock/Turbulent Boundary Layer/Bleed Interactions," AIAA Paper 92-3085, July 1992.

10. Hamed, A., Shih, S.H., and Yeuan, J.J., "A Parametric Study of Bleed in Shock Boundary Layer Interactions," AIAA Paper 93-0294, Jan. 1993.

11. Hamed, A., Yeun, J.J., and Shih, S.H., "An Investigation of Shock Wave Turbulent Boundary Layer Interactions with Bleed through Slanted Slots," AIAA Paper 93-2992, July 1993.

12. Hahn, T.O., Shih, T.I-P., and Chyu, W.J., "Numerical Study of Shock-Wave/Boundary-Layer Interactions with Bleed," AIAA Journal, Vol. 31, No. 5, 1993, pp. 869-876.

13. Rimlinger, M.J., Shih, T.I-P., and Chyu, W.J., "Three-Dimensional Shock-Wave/Boundary-Layer Interactions with Bleed Through a Circular Hole," AIAA Paper 92-3084, July 1992.

14. Shih, T.I-P., Rimlinger, M.J., and Chyu, W.J., "Three-Dimensional Shock-Wave/Boundary-Layer Interaction with Bleed," AIAA Journal, Vol. 31, No. 10, 1993, pp. 1819-1826.

15. Chyu, W.J., Rimlinger, M.J., and Shih, T.I-P., "Effects of Bleed-Hole Geometry and Plenum Pressure on Three-Dimensional Shock-Wave/Boundary-Layer/Bleed Interactions," AIAA Paper 93-3259, July 1993.

16. Rimlinger, M.J., Shih, T.I-P., and Chyu, "Three-Dimensional Shock-Wave/Boundary-Layer Interactions with Bleed through Multiple Holes," AIAA Paper 94-0313, January 1994.

17. Benek, J.A., Buning, P.G., and Steger, J.L., "A 3-D Chimera Grid Embedding Technique," AIAA Paper 85-1523, 1985.

18. Benek, J.A., Donegan, T.L., and Suhs, N.E., "Extended Chimera Grid Embedding Scheme with Application to Viscous Flows," AIAA Paper 87-1126, 1987.

19. Buning, P.G. and Chan, W.M., "OVERFLOW/F3D User's Manual," NASA - Ames Research Center, 1991.

20. Baldwin, B. and Lomax, H., "Thin Layer Approximation and Algebraic Model for Separated Turbulent Flows," AIAA Paper 78-257, 1978.

21. Huang, P.G., Bradshaw, P., and Coakley, T.J., "Skin Friction and Velocity Profile Family for Compressible Turbulent Boundary Layers," AIAA Journal, Vol. 31, No. 9, 1993, pp. 1600-1604.

22. Steger, J.L. and Warming, R.F., "Flux-Vector Splitting of the Inviscid Gasdynamic Equations with Application to Finite-Difference Methods," Journal of Computational Physics, Vol. 40, No. 2, 1981, pp. 263-293.

23. Steger, J.L., Ying, S.X., and Schiff, L.B., "A Partially Flux-Split Algorithm for Numerical Simulation of Compressible Inviscid and Viscous Flow," Proceedings of the Workshop on Computational Fluid Dynamics, Institute of Nonlinear Sciences, University of California, Davis, California, 1986.

24. Walatka, P.P. and Buning, P.G., "PLOT3D User's Manual," NASA - Ames Research Center, 1989.

## APPENDIX
## DESCRIPTION OF INPUTS INTO AUTOMAT

As mentioned in the section on User Interface, for interactive sessions, the user will be prompted on the screen for the input data. For non-interactive sessions, the inputs are all contained in a namelist file, which can be modified by the user via any text editor (e.g., vi in unix). Also mentioned was that the namelist file required for non-interactive sessions is automatically generated after every interactive session to minimize effort needed to input data for future sessions. In this appendix, the input file, namelist.inp (default name given by AUTOMAT to the input file) is described in detail. A typical namelist.inp file is as follows:

```
no   ! Not an interactive session
 $flow
   metric = .T. , fsmach =  1.60 , pinf =  61803.00 , tinf = 198.41,
   rgas = 287.200 , redl = 0.37130E+08 , momthick = 0.91400E-03 ,
   shock = .T. , thetwdge =  7.50 ,
   $end
 $pattern
   axmaj =  0.2540E-02 , axmin =  0.2540E-02 , nhole =  4 ,
   pitchx =  0.8799E-02 , pitchy =  0.5080E-02 , halpha =  90.00 ,
   depth =  0.1016E-01 , shockx = 0.17598E-01 ,
   $end
 $holepar
   finrad =  0.25 , fover =  3.00,
   dhmao =  0.5E-04 , dhmai =  0.1E-03 ,
   dhmio =  0.5E-04 , dhmii =  0.1E-03 ,
   delhent =  0.1495E-05 , delhmid =  0.2E-03 , delhexit =  0.5E-04 ,
   darc =  0.1814E-03 ,
   $end
 $platepar
   finfx =  4.00 , foutx =  2.00 , ferri =  1.00 , ferro =  1.0 ,
   fskirtu =  2.00 , fskirtd =  2.00 , finfz =  6.00 ,
   din =  0.1270E-02 , dout =  0.7620E-03 , dbl =  0.6226E-03 ,
   delstream =  0.3E-03 , delspan =  0.1E-03 ,
   $end
 $plenumpar
   flwall =  1.00 , finfz =  5.00 ,
   dlw =  0.8E-04 , dbw =  0.5E-04 ,
   dmidz =  0.1270E-02 , delhexit =  0.5E-04 ,
   $end
 $overlapar
```

```
 ntop =  4 , nbot = 10 ,
 $end
$backpress
 prback = 0.3,
 $end
```

An explanation of each parameter in the above namelist is given below.

- First line should always have the word "no". This alerts the program to use a non-interactive session.

- $flow: Parameters related to flow conditions.

  metric: T implies SI units, and F implies English units. All parameters having a numerical value should be given in the units chosen here.
  fsmach: freestream mach number at inflow ($M_\infty$)
  pinf: pressure at inflow ($P_\infty$)
  tinf: temperature at inflow ($T_\infty$)
  rgas: gas constant for fluid
  redl: reynolds number per unit length
  momthick: momentum thickness of boundary layer ($\theta$)
  shock: T or F indicating the presence of an incident shock wave
  thetwdge: half-angle in degrees of shock generating wedge ($\beta$)

- $pattern: Parameters related to bleed holes.

  axmaj: radius of hole on major axis (if bleed hole is inclined, then it appears as an ellipse on the flat plate; for normal holes, axmaj = axmin = D, D = diameter of hole)
  axmin: radius of hole on minor axis (same comment as that for axmaj)
  nhole: total number of holes (N)
  pitchx: spacing between hole centers in x (streamwise) direction ($L_x$)
  pitchy: spacing between hole centers in y (spanwise) direction ($L_y$)
  halpha: angle between bleed hole centerline and flat plate ($\alpha$)
  depth: thickness of plate ($L_t$)
  shockx: location of where the incident shock wave impinges on the flat plate under inviscid conditions measured from the center of the first bleed hole (i.e., the origin of the coordinate system for the final grid)

- $holepar: Parameters used in generating the bleed hole grids

  finrad: the inner radius of the O-H bleed hole grid is given as $r_{imajor} = r_{omajor}$*finrad = axmaj*finrad
  fover: determines the overlap of the H-H and O-H grids in each bleed hole (fover specifies how many grid lines will be overlapped)
  dhmao: O-H grid spacing in radial direction at outer edge of major axis
  dhmai: O-H grid spacing in radial direction at inner edge of major axis
  dhmio: O-H grid spacing in radial direction at outer edge of minor axis
  dhmii: O-H grid spacing in radial direction at inner edge of minor axis
  delhent: spacing in z (depth) direction at the entrance of both hole grids
  delhmid: spacing in z (depth) direction at the midplane of both hole grids
  delhexit: spacing in z (depth) direction at the exit of both hole grids
  darc: O-H grid is equally-spaced in the circumferential direction (darc is the arclength of that spacing)

- $platepar: Parameters used in generating the flat plate grid

finfx: used to specify the x-location of the inflow

Procedure is as follows:
- Specify the height at which the shock wave enters the computational domain. In order to determine this height, choose it as some multiple of the boundary layer height at the inflow. $z_{shockinflow}$ = **finfx***blthick
- Calculate the x position of the inflow plane based upon the equation of the straight line describing the impinging shock wave.

foutx: used to specify the x-location of outflow (same procedure are above; $z_{shockoutflow}$ = **foutx***blthick)

Note: If there is no shock wave present, finfx and foutx represent the number of bleed hole diameters upstream of first bleed hole and downstream of last bleed hole to place the inflow and outflow planes; $x_{inflow}$ = (finfx)*2*axmaj.

ferri, ferro: error contingencies for locating the inflow and outflow planes

In cases where the incident or reflected shocks are steep, the inflow and outflow boundaries might be located too close to the bleed holes. The locations are then forced to be as follows:
- $x_{inflow}$ = (fskirt + **ferri**)*2*axmaj
- $x_{outflow}$ = (fskirt + **ferro**)*2*axmaj

skirtu, determines the size of the equi-spaced region upstream and downstream of the bleed hole

fskirtd: pattern; $x_{up\text{-}eqsui\text{-}start}$ = $x_{hole1}$ - **fskirtu***2*axmaj

finfz: determines the height of the domain; $z_{top}$ = finfz*blthick (the code checks to ensure that the shock inflow position and the domain height are compatible. i.e. finfz > finfx)

din: spacing in streamwise direction at inflow plane

dout: spacing in streamwise direction at outflow plane

dbl: spacing at the top of the boundary layer (there is an equally-spaced region from z = blthick to 2*blthick)

delstream: spacing in streamwise direction above the bleed hole pattern

delspan: spacing in the spanwise direction.

- $plenmpar: Parameters controlling plenum grid generation

flwall: location of left wall in plenum; x = (flwall+fskirtu)*2*axmaj

finfz: location of lower wall in plenum; z = -(depth + finfz*axmaj*2)

dlw: spacing at left wall

dbw: spacing at bottom wall

dmidz: spacing in z direction at location halfway between the exit of the hole and the bottom wall

delhexit: spacing in the z direction at the top wall (matches the spacing at the outflow of the hole)

- $overlapar: Parameters governing the overlapped regions above the plate and into the plenum of the hole grids.

ntop: number of grid lines to extend the hole grids above the plate

nbot: number of grid line to extend the hole grids in the plenum

- $backpress

prback: ratio of back pressure to freestream pressure ($P_b/P_\infty$)

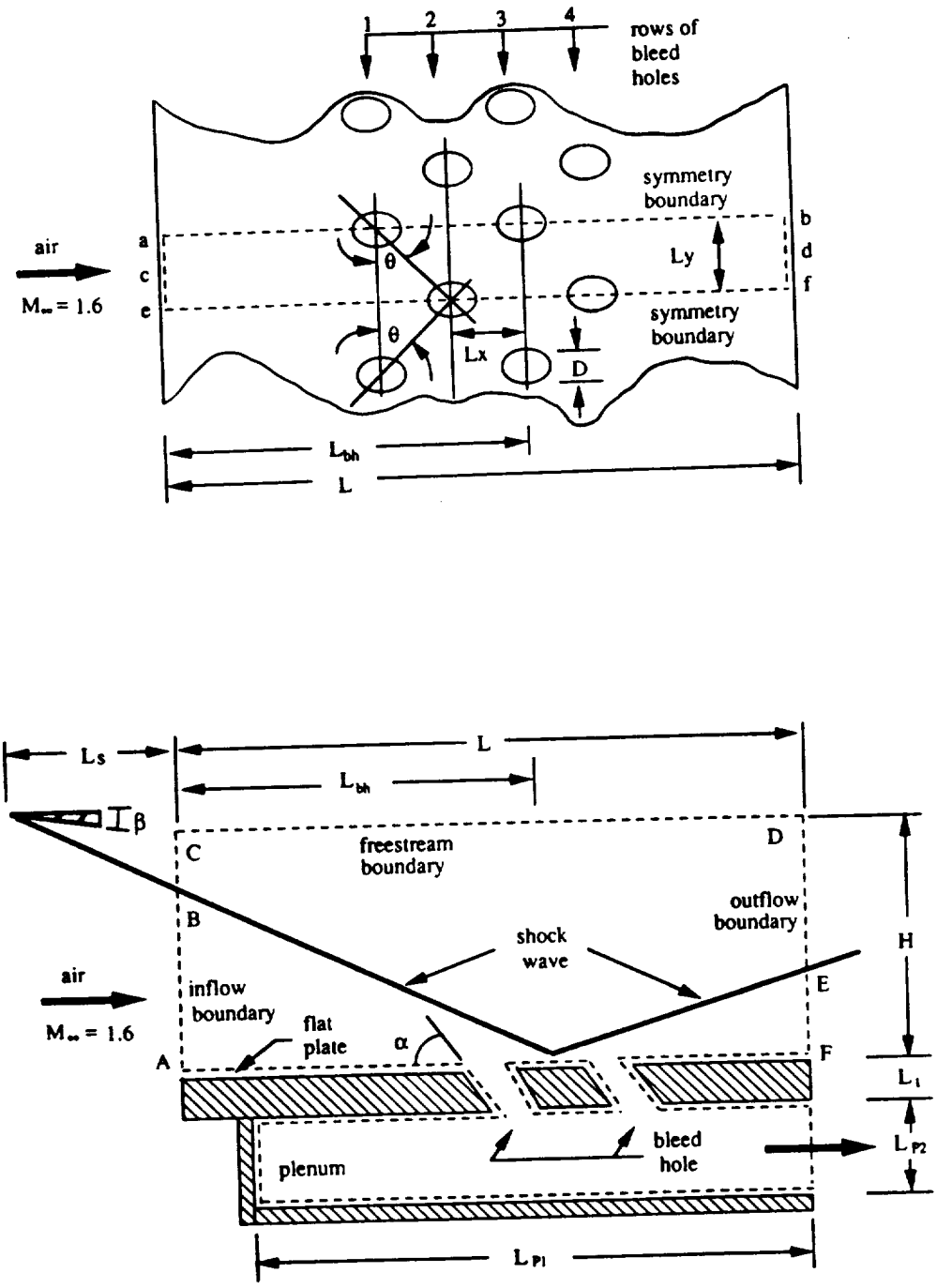



Figure 1 Schematic of a typical inlet-bleed problem with four rows of bleed holes.

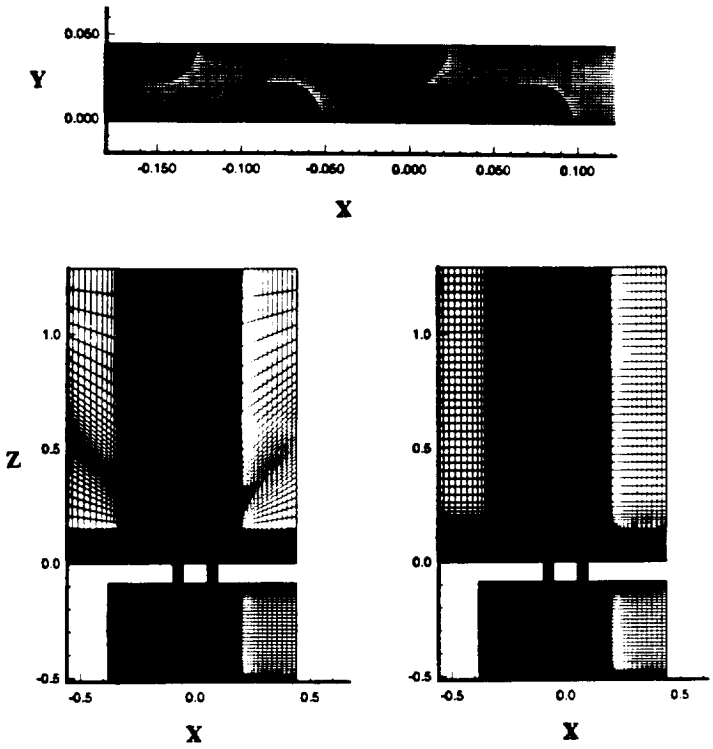



Figure 2 Typical grid system generated by AUTOMAT.

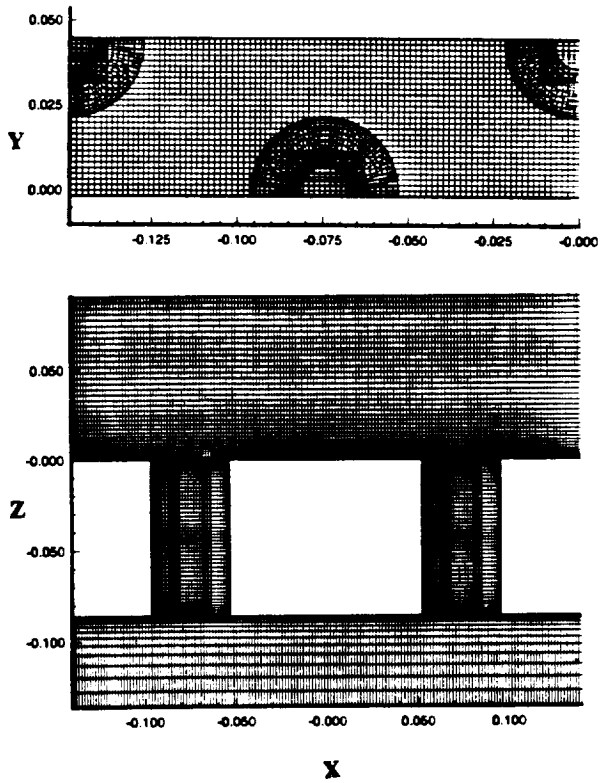



Figure 3 Closeup view of grid system near a bleed hole.

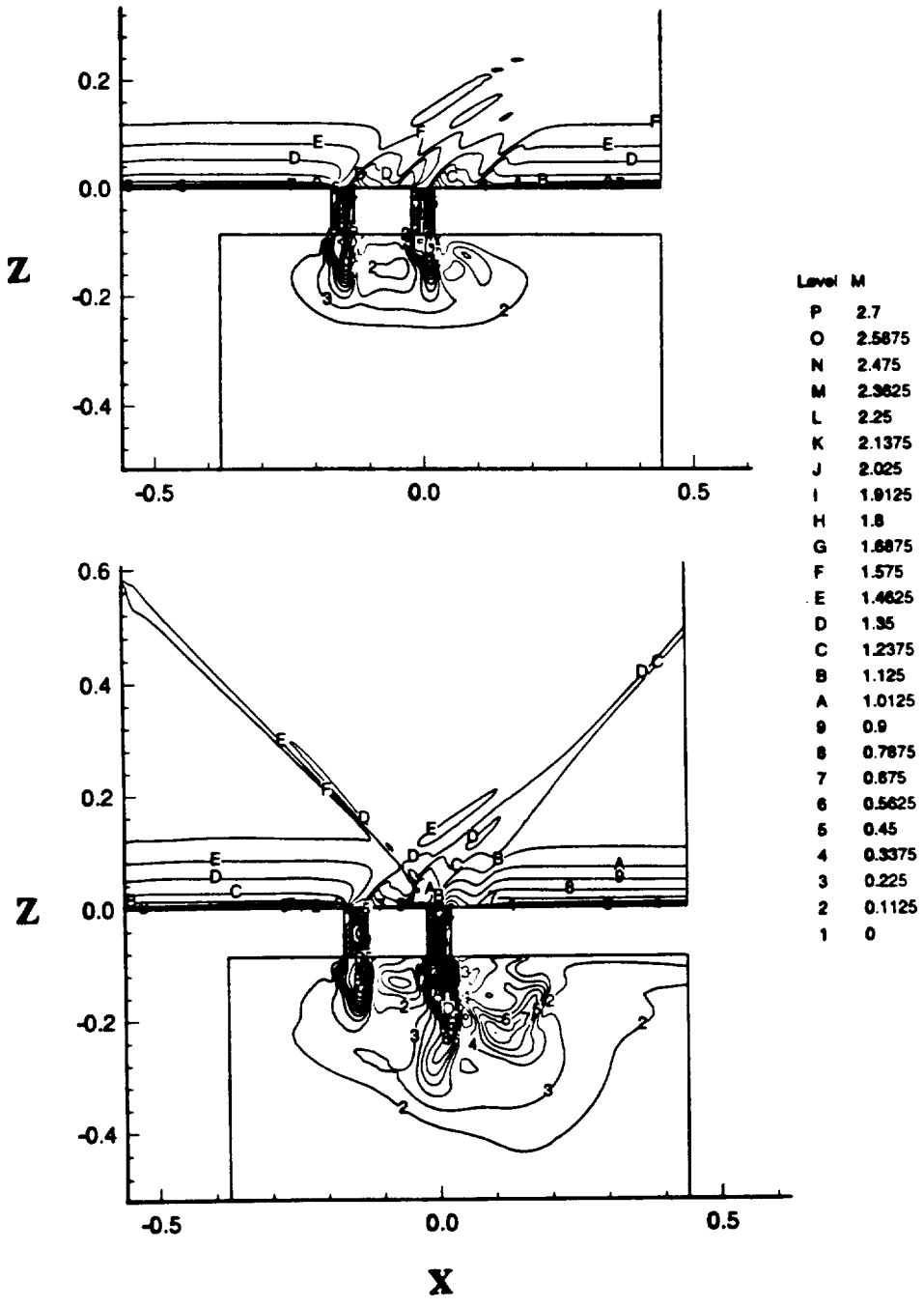



Figure 4 Typical solution (Mach number contours) generated by OVERFLOW with grid generated by AUTOMAT. Top figure: without incident shock. Bottom figure: with incident shock.

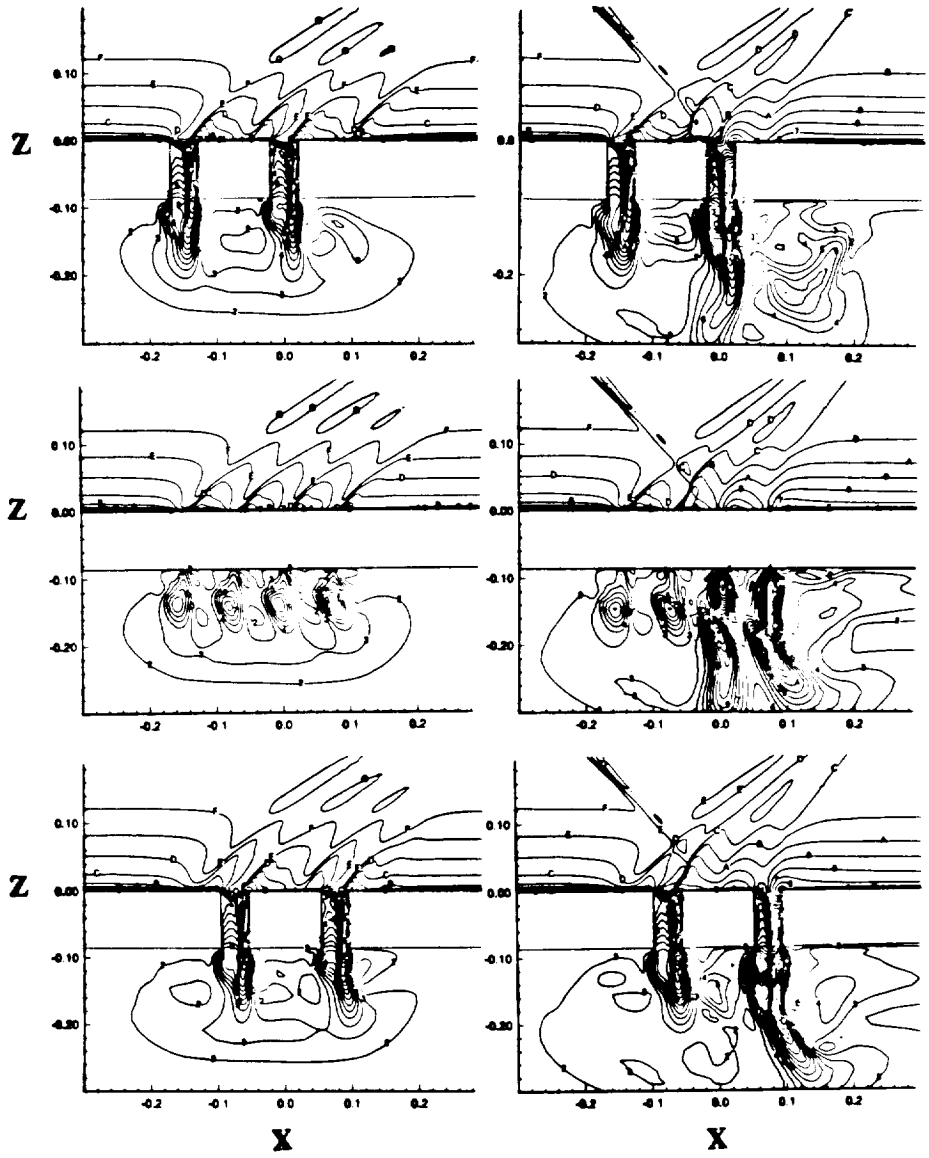



Figure 5 Typical solution (Mach number contours near a bleed hole) generated by OVERFLOW with grid generated by AUTOMAT. Left: without incident shock along three cross sections (top: a-b, middle: c-d, bottom: e-f). Right: with incident shock along three cross sections (top: a-b, middle: c-d, bottom: e-f). See Fig. 1 for definition of a-b, c-d, and e-f cross sections.

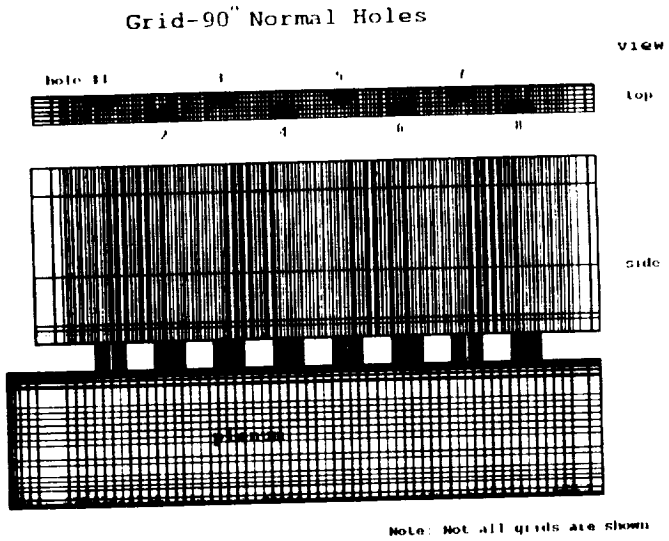



Figure 6 Grid system generated for an inlet-bleed problem with eight rows of normal bleed holes arranged in a staggered fashion (not all grid points shown).

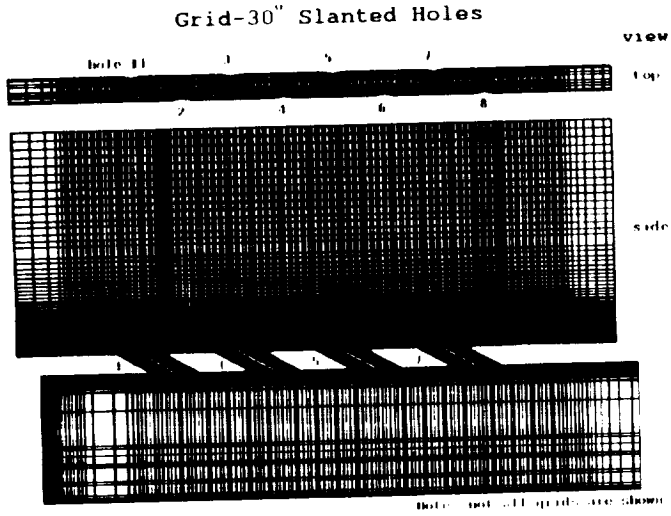



Figure 7 Grid system generated for an inlet-bleed problem with eight rows of slanted bleed holes arranged in a staggered fashion (not all grid points shown).

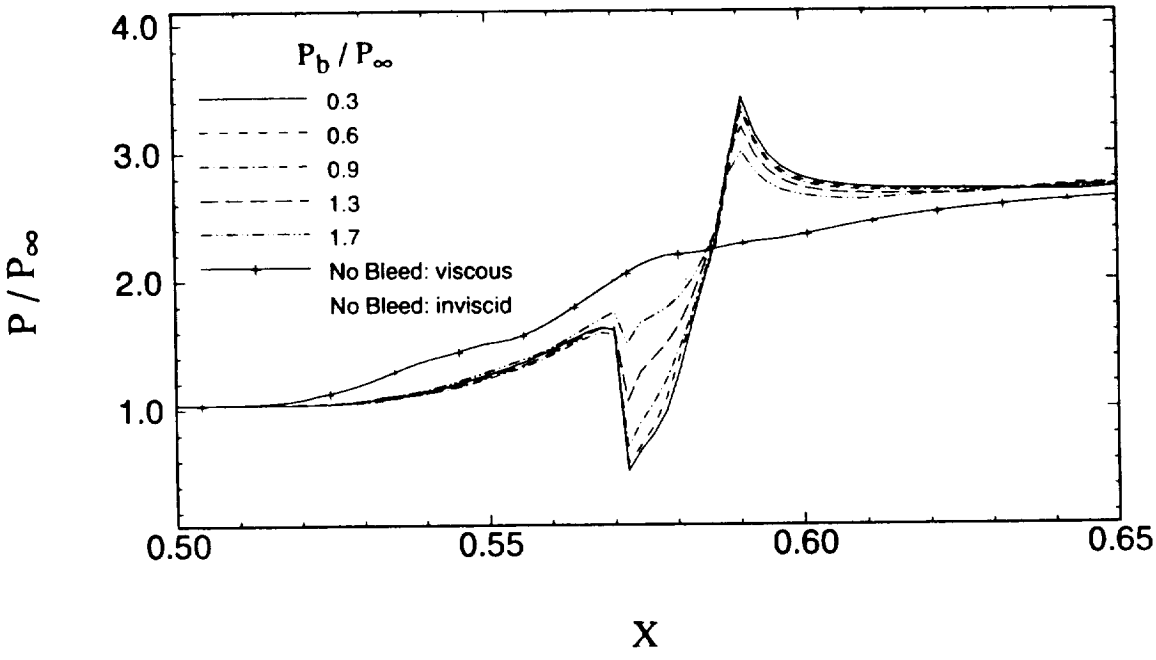



Figure 8 Pressure along surface of flat plate from inflow to outflow passing through center of bleed hole for a single 90° bleed hole.

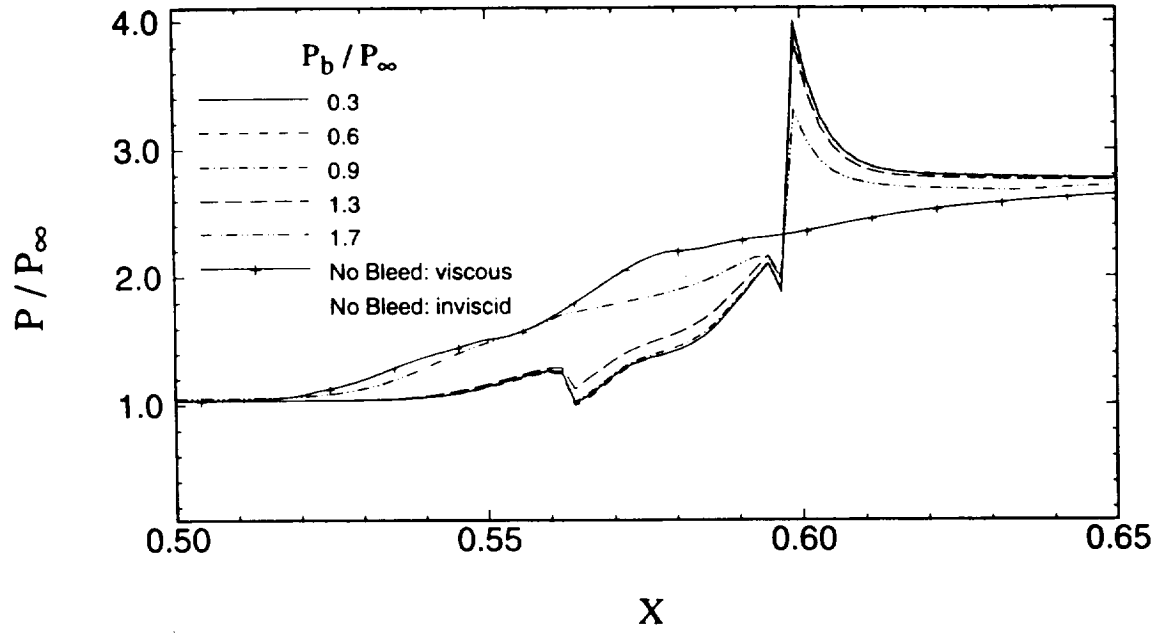



Figure 9 Pressure along surface of flat plate from inflow to outflow passing through center of bleed hole for a single 30° bleed hole.

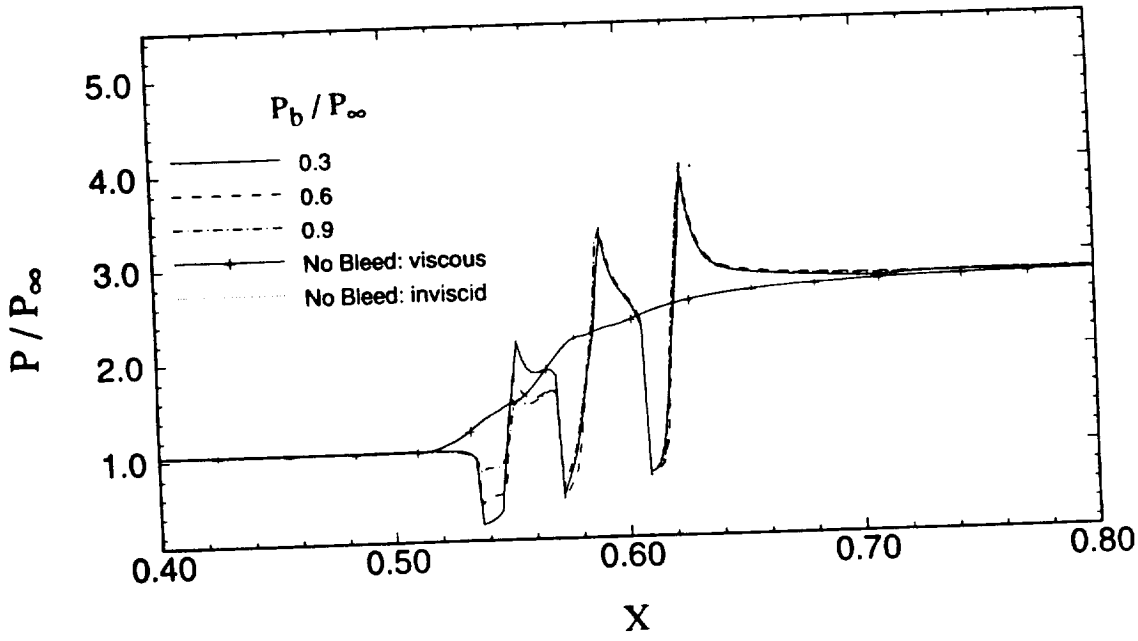



Figure 10 Pressure along wall from inflow to outflow passing through center of bleed holes for three 90° holes.

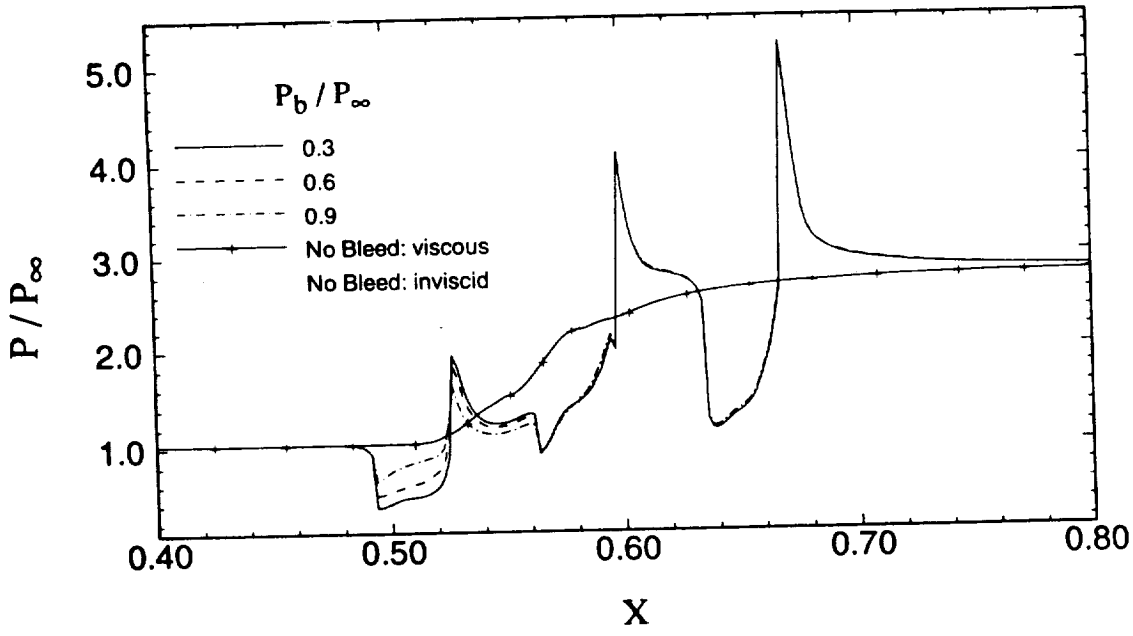



Figure 11 Pressure along wall from inflow to outflow passing through center of bleed holes for three 30° holes.

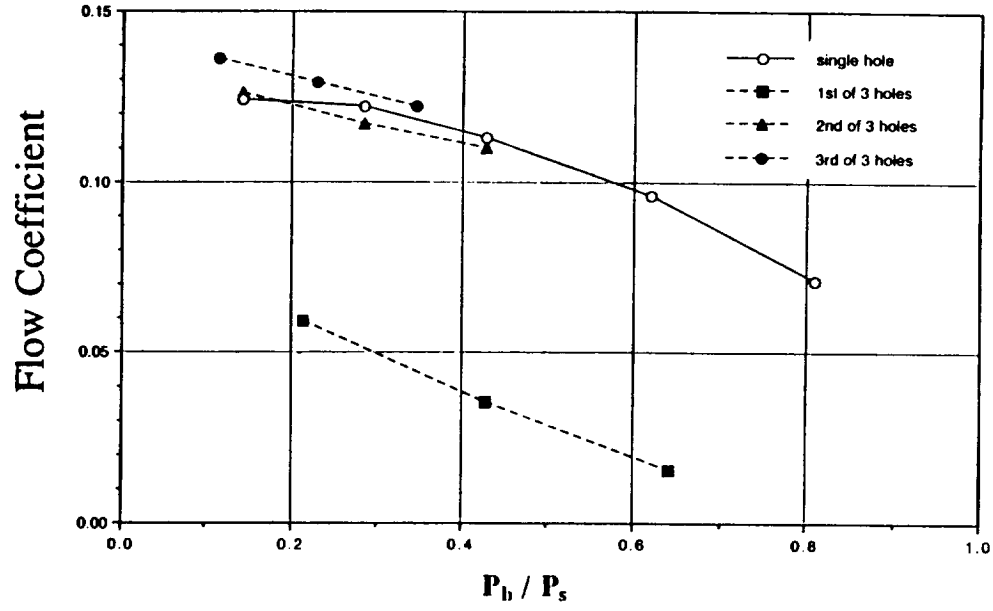



Figure 12 Flow coefficient as a function of $P_b/P_s$ for 90° bleed hole or holes.

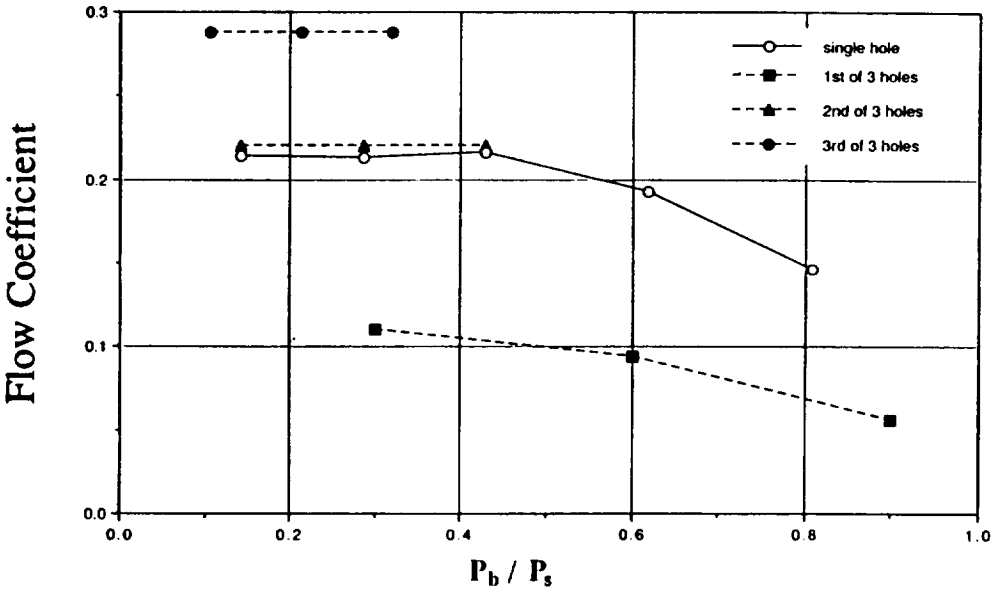



Figure 13 Flow coefficient as a function of $P_b/P_s$ for 30° bleed hole or holes.

N95-28769

# A TECHNIQUE FOR OPTIMIZING GRID BLOCKS

John F. Dannenhoffer, III*

United Technologies Research Center
East Hartford, CT 06108

## ABSTRACT

A new technique for automatically combining grid blocks of a given block-structured grid into logically-rectangular clusters which are "optimal" is presented. This technique uses the simulated annealing optimization method to reorganize the blocks into an optimum configuration, that is, one which minimizes a user-defined objective function such as the number of clusters or the differential in the sizes of all the clusters. The clusters which result from applying the technique to two different two-dimensional configurations are presented for a variety of objective function definitions. In all cases, the automatically-generated clusters are significantly better than the original clusters. While this new technique can be applied to block-structured grids generated from any source, it is particularly useful for operating on block-structured grids containing many blocks, such as those produced by the emerging automatic block-structured grid generators.

## INTRODUCTION

Traditionally, the block-structured grid generation process has been called "complete" when a set of blocks and grid points which provide adequate resolution for a given configuration has been defined. While the location of the grid points is usually carefully controlled to provide the required spacings and grid quality, the location of the block boundaries is chosen arbitrarily to aid in the grid generation process.

Unfortunately, the location of the block boundaries can have a strong influence on the overall performance of the flow solver, especially when employing one of today's advanced computer architectures. For example, for a vector computer one typically wants as few blocks as possible, while simultaneously keeping the blocks as "long" as possible in order to maximize use of the vector pipelines. Alternatively, for a parallel computer (or a network of workstations) one typically wants the blocks to be as load-balanced as possible in order to minimize idle time. To compound the problem, the computer architecture which will be employed to obtain the flow solution is not always be known at grid generation time. Therefore, it is desirable to have a procedure which can recombine blocks (that is, create clusters) of any block-structured grid, regardless of the software used to generate it.

In addition, a new class of block-structured grid generators is emerging which "automatically" generates block-structured grids for a variety of configurations[1, 2]. A general property of the grids which these procedures generate is that they produce a very large number of grid blocks, sometimes running into the

*Senior Research Engineer



PRECEDING PAGE BLANK NOT FILMED

C-9

tens of thousands. This new clustering procedure could be used in such cases to significantly reduce the number of clusters (blocks) on which the flow solver would have to operate.

The objective of this work is to develop a general technique for automatically combining blocks of a given block-structured grid into logically-rectangular "clusters" so as to minimize a user-defined objective function.

## TECHNICAL APPROACH

### The Clustering Problem

Consider the block-structured grid configuration shown in Figure 1. It consists of seven grid blocks (the seven squares labeled A through G) which are arranged so as to fill an L-shaped domain; also shown are eight "removable" edges (which are labeled 1 through 8).

Simply stated, the objective in the clustering problem is to find the set of edges which, when removed, yields both a "valid" configuration and which minimizes some "score". For now, assume that the "score" is the number of clusters. A "valid" clustering is one which solely consists of (logical) rectangles; L-shaped clusters are invalid.

One could reduce the score of the clustering in Figure 1 by removing any edge; for example, when edge 7 is removed, the clusters shown in Figure 2 result, with the score now reduced to the better value of 6. To further reduce the score, either edge 1, 2, 3, 4, or 6 can be removed to yield a valid clustering and a reduced score of 5. But removal of either edge 5 or 8 yields a cluster (for example, cluster EFG in Figure 3) which is not logically-rectangular and thus invalid. This process can be continued, yielding the optimal score is 2 which is produced either by retaining only edge 2 or by retaining only edges 3 and 4.

### Optimization Procedure

The optimization problem considered here is one which is characterized as a combinatorial minimization problem. That is to say, the design space cannot be characterized as an N-dimensional surface defined over N smoothly-varying design parameters. Rather it is a design space which is described by a set possible configurations. Other problems of this combinatorial nature include VLSI circuit layout[3] and the infamous traveling salesman problem[4].

There are two major difficulties associated with combinatorial optimization problems which do not arise in more traditional optimization problems. First, the design space can become extremely large because the configurations generally are describable by a sequence of events (the number of which, when taken together, grows factorially). Second, because the design variables can only take on discrete values (such as "on" and "off"), any procedure which exploits the concept of gradient is not relevant.

One technique which has proven very successful at solving the combinatorial optimization problem is the simulated annealing algorithm, which was first proposed by Metropolis[5] and which is described extensively by Aarts and Korst[6].

Stated very briefly, the simulated annealing algorithm is: starting from a known valid configuration, randomly select a change of state; any change which results in lower objective function value or which results in an objective function value which is not "too much worse" than that of the previous configuration is accepted. This process is repeated with the definition of "too much worse" tightening as it proceeds, until no proposed changes to the configuration yield any reduction in the objective function.

This procedure gets its name from an analogy with the thermodynamic process of annealing, or freezing, of liquids into solids. In the beginning stages of the solidification process, molecules move freely in an attempt to form crystalline structures which have low internal energies (the objective function). For annealing, the temperature is slowly reduced and the mobility of the molecules is gradually restricted, resulting in minimum energy crystals. Alternatively, for quenching, the temperature is quickly reduced, resulting in a crystalline structure which has significantly higher energy.

These same concepts can be applied to optimization. The initial steps of annealing (when the control parameter is high) is similar to a random-walk procedure, which is known to be an effective strategy for getting into the neighborhood of a global optimum. The final steps of annealing (when the control parameter is low) is similar to a hill-climbing procedure, which is known to be effective in the vicinity of an optimum. The annealing schedule provides an orderly mechanism of transitioning from one to the other. It should be noted that quenching is analogous to hill-climbing alone, which is notorious for getting stuck in local extrema (as are all greedy algorithms).

In terms of pseudo-code, the algorithm can be stated as follows:

* initialize the configuration and compute its objective function $\mathcal{O}$

* set an initial value for the control parameter $T$

* do { (outer loop of generations)

  * do { (inner loop of attempts)

    * propose a random change to the configuration

    * compute the change in objective function $\Delta\mathcal{O}$ of the proposed change

    * if ($\Delta\mathcal{O} < 0$ or random$(0 \rightarrow 1) < e^{-\Delta\mathcal{O}/T}$) then

      - take the step and adjust $\mathcal{O}$

    } until (stop criterion is met)

  * decrease the control parameter $T$

  } until (there were no successful proposals at the previous $T$)

In order to use this algorithm for optimizing block clusters, one needs to provide the following:

**Configuration** (a method of describing the current "state" of a configuration). For the current application, the current configuration can be described as a table that says which edges currently exist (or conversely, which edges have been removed);

**Rearrangements** (a method for proposing and carrying out changes to the configuration). Here the procedure is to pick an edge at random, and if it exists, remove it; alternatively if it has already been removed, then re-insert it. In this way, the effects of any previous rearrangements can be undone (which, by the way, is generally an important property when using simulated annealing);

**Objective function computation** (an efficient procedure for computing the change in objective function given any proposed change to the configuration). One of the nice features of using the simulated annealer is that, except for the initial configuration, one only has to calculate the change in objective function $\Delta\mathcal{O}$ that results from a proposed configuration change. This is indeed fortunate, because for the clustering problem, the change in objective function can be computed in a small, fixed amount of work (that is, independent of the problem size), whereas the work to compute the actual objective function scales with $N$ (the number of removable edges in the configurations); and

**Annealing schedule** (a recipe for controlling the decrease of the control parameter $T$). This is the least rigorous part of the simulated annealing algorithm. In general, a suitable annealing schedule can only be found by experimenting.

## Objective Function Formulation

In addition to the selection of suitable "design variables" (discussed above), the application of the simulated annealing algorithm to the clustering problem requires the definition of the "objective function" and "constraints".

First consider the objective function, $\mathcal{O}$. The most obvious choices include:

**Minimize number of clusters** This objective function is appropriate when one wants to minimize the number of concurrent processors which are needed to perform some calculation. A suitable form might be

$$\mathcal{O} = N_{\text{clus}}$$

where $N_{\text{clus}}$ is the number of clusters.

**Minimize size of largest cluster** When running a calculation on a parallel processor or on a group of workstations, one typically wants to distribute the work as evenly as possible across the processors. A suitable objective function might be

$$\mathcal{O} = N_{\text{size}}$$

where $N_{\text{size}}$ is the size of (number of grid points in) the largest cluster. (Note that by itself, this objective function might arrive at the solution that each original cluster should remain unchanged.)

**Hybrid** Clearly a hybrid objective function could be formed which combines the above, with suitable coefficients, as in

$$\mathcal{O} = \lambda_{\text{clus}} \min\left(N_{\text{clus}}, L_{\text{clus}}\right) + \lambda_{\text{size}} \min\left(N_{\text{size}}, L_{\text{size}}\right)$$

where $\lambda_{\text{clus}}$ and $\lambda_{\text{size}}$ are weighting coefficients and $L_{\text{clus}}$ and $L_{\text{size}}$ are user-specified limits. These limits are particularly useful for tuning the objective function in a variety of ways. For example, one could minimize the number of clusters with the restriction that no cluster to be larger than some specified size (as might be imposed by a memory restriction in some processor). Similarly, given the number of clusters, one could minimize the size of the largest cluster (in essence, this amounts to load-balancing).

Notice that in the hybrid formulation of the objective function, the "weak" constraints imposed by $L_{\text{size}}$ and $L_{\text{clus}}$ are actually written as part of the objective function. There is in addition a "strong" constraint on this problem, namely the requirement that all clusters be logically rectangular. This can be accounted for by adding another "penalty" term to the objective function, or

$$\mathcal{O} = \lambda_{\text{clus}} \min\left(N_{\text{clus}}, L_{\text{clus}}\right) + \lambda_{\text{size}} \min\left(N_{\text{size}}, L_{\text{size}}\right) + \lambda_{\text{brak}} N_{\text{brak}}$$

where $N_{\text{brak}}$ is the number of edges which would have to be re-inserted to yield all logically-rectangular clusters. In general, $\lambda_{\text{brak}} > \lambda_{\text{clus}}$ to ensure that the final clusters are all well shaped.

## COMPUTED RESULTS

As demonstration cases, results were obtained on several two-dimensional configurations; two of them are shown here.

The first test case is for an internal-flow passage containing eight circular posts (hence the test case is called "posts"). The original grid for this case, which contains 5543 nodes in 89 clusters (each block was initially assigned to its own cluster), is shown in Figure 4a.

For the first experiment on the "posts" test case, the simulated annealer was applied to minimize the number of clusters (the user-supplied constants are given in Table I). Intermediate results, that is the clustering at the end of each setting of the control parameter $T$, are shown in Figures 4b-e. Notice that most of these intermediate results contain many badly-shaped clusters (that is, they are not logically rectangular). The final clustering, which is shown in Figure 4f, contains 19 clusters, where the largest cluster contains 969 nodes. Table I summarizes these intermediate and final results.

For this case, as well as all others shown here, the control parameter was initially set to $T = 5.0$ and was reduced at the end of each generation (the outer loop) by a factor of $T_{\text{new}}/T_{\text{old}} = 0.75$; these values, which were determined through numerical experimentation, have been found to be sufficient for all test cases executed to date. Also, the stopping criterion at the end of the attempts (inner) loop was taken to be the first of: $10N$ successful attempts or $100N$ total attempts, whichever comes first (where $N$ is the number of removable edges). Again, these values were determined by numerical experiments and were found to work well for all cases which have been tried to date.

In order to better understand how the simulated annealer works, a convergence history for this case was plotted in Figure 5. The abscissa represents the successful attempt number while the ordinate shows the current value of the objective function $\mathcal{O}$. The circles indicate those times when the control parameter $T$ is decreased. There are a number of interesting features evident in the Figure:

- the objective function $\mathcal{O}$ does not strictly decrease, but rather tends toward an optimum;
- there is a "noise band" corresponding to the concept of "not too much worse". As expected, the width of the band decreases as the control parameter decreases; and
- the number of successes between control parameter reductions decreases, corresponding to the fact that as $T$ gets smaller, it is more difficult to find a success.

It should be noted that the simulated annealing algorithm is stochastic because of the random number generator used both to select a proposed change and in the acceptance check. In theory then, the results which it produces cannot necessarily be repeated. In order to determine the sensitivity of the present results to the random number seed, a few of the test cases have been re-executed many times. In all cases, the "optimum" solutions were very close to each other and were very far from the initial solution.

One further point should be made about the algorithm. The work required to reach "convergence" is bounded by $N \times M$, where $N$ is the number of design variables (the number of removable edges) and $M$ is the number of times which the control parameter $T$ needs to be decreased. Although in the worst case $M \to \infty$, test cases of varying sizes seem to indicate the $M$ is approximately constant, with a value of about $M \approx 10$. In fact, even though in the current implementation a limit of $M = 20$ has been hard-coded (to protect against infinite looping), it has never been exercised.

For the second experiment on the "posts" test case, the number of clusters were again minimized, but with an upper limit $L_{\text{size}} = 500$ on the size of any one cluster. The clustering which results for this case, which is shown in Figure 6a and is summarized in Table I, is very different from that discussed above.

A third experiment on the "posts" test case was conducted, wherein the size of the largest cluster was to be minimized with the constraint the the number of clusters be bounded by $L_{\text{clus}} = 25$. The results of

this test are shown in Figure 6b and again in Table I. Again the clusters which are formed are different from the above two cases, and in general not intuitively obvious.

A second test case, corresponding to a gas-turbine combustor cross-section (and hence call "combustor") is also shown here. The Figures 7 and 8 and Table II summarize its results. The results, which are very similar to those obtained for the "posts" test case, serve to demonstrate the broad applicability of the new clustering technique. As in the "posts" test case, all cases executed in less than two minutes of CPU time.

## CONCLUSIONS

A new procedure for optimizing "clusters" of grid blocks has been described. The procedure, which uses the simulated annealing optimization algorithm, has been executed on many configurations; it has been demonstrated here on two different two-dimensional configurations. In all cases, the new algorithm has yielded clusterings which are "good" in the sense that they have either significantly reduced the total number of clusters (with and without a limit on the maximum cluster size) or have balanced the cluster sizes (with a given number of clusters). Even though the simulated annealer does not guarantee that an actual global extremum has been achieved, the results of the automatic algorithm were at least as good as the clusterings which were manually generated. All cases executed required less than two CPU minutes on an Silicon Graphics Indigo2-R4400 for problem sizes of about 100 removable edges; numerical experiments indicates that the time should scale linearly with problem size. While this procedure is applicable to block-structured grids generated by any software, it is particularly well-suited to those generated by the emerging "automatic" block-structured grid generators.

## ACKNOWLEDGEMENTS

This work was funded under the UTC Corporate Sponsored Research Program. The author would like to thank David Sirag for his helpful insights and comments about the simulated annealing algorithm as well as many of his other colleagues who reviewed this manuscript.

## REFERENCES

[1] Dannenhoffer, JF, "Automatic Blocking for Complex 3-D Configurations," NASA CP-3291, May 1995.

[2] Shaw, JA, and Weatherill, NP, "Automatic Topology Generation for Multiblock Grids," *Applied Math. and Comp.*, vol 53, pp 355–388, 1992.

[3] Vecchi, MP, and Kirkpatrick, S, "Global wiring by simulated annealing," *IEEE Trans. on Computer-Aided Design*, vol 2, pp 215-222, 1983.

[4] Aarts, EHL, Korst. JHM, and VanLaarhoven, PJM, "A quantitative analysis of the simulated annealing algorithm: a case study for the traveling salesman problem," *J. of Statistical Physics*, vol 50, pp 189-206, 1988.

[5] Metropolis, N, Rosenbluth, A, Teller, A, and Teller, E, *J. Chem Phys.*, vol 21, p. 1087, 1953.

[6] Aarts, E and Korst, J, *Simulated Annealing and Boltzmann Machines*, Wiley & Sons, 1989.

| Figure | $\lambda_{clus}$ | $L_{clus}$ | $\lambda_{brak}$ | $\lambda_{size}$ | $L_{size}$ | $N_{clus}$ | $N_{brak}$ | $N_{size}$ |
|---|---|---|---|---|---|---|---|---|
| 4a | 1.00 | 1 | 2.00 | 0.001 | 0 | 89 | 0 | 425 |
| 4b | | | | | | 23 | 28 | 1666 |
| 4c | | | | | | 22 | 26 | 1543 |
| 4d | | | | | | 28 | 16 | 1027 |
| 4e | | | | | | 31 | 13 | 850 |
| ⋮ | | | | | | | | |
| 4f | | | | | | 19 | 0 | 969 |
| 6a | 1.00 | 1 | 2.00 | 0.100 | 500 | 23 | 0 | 455 |
| 6b | 1.00 | 25 | 2.00 | 0.001 | 0 | 25 | 0 | 441 |

Table I: Summary of results for test case "posts".

| Figure | $\lambda_{clus}$ | $L_{clus}$ | $\lambda_{brak}$ | $\lambda_{size}$ | $L_{size}$ | $N_{clus}$ | $N_{brak}$ | $N_{size}$ |
|---|---|---|---|---|---|---|---|---|
| 7a | 1.00 | 1 | 2.00 | 0.001 | 0 | 80 | 0 | 273 |
| 7b | | | | | | 30 | 19 | 633 |
| 7c | | | | | | 31 | 17 | 978 |
| 7d | | | | | | 21 | 14 | 1356 |
| 7e | | | | | | 36 | 4 | 533 |
| ⋮ | | | | | | | | |
| 7f | | | | | | 16 | 0 | 828 |
| 8a | 1.00 | 1 | 2.00 | 0.100 | 500 | 22 | 0 | 468 |
| 8b | 1.00 | 25 | 2.00 | 0.001 | 0 | 25 | 0 | 468 |

Table II: Summary of results for test case "combustor".

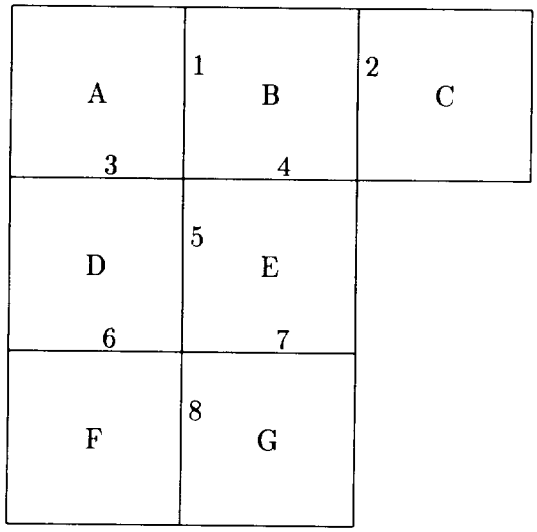



Figure 1: Original configuration. Score=7.

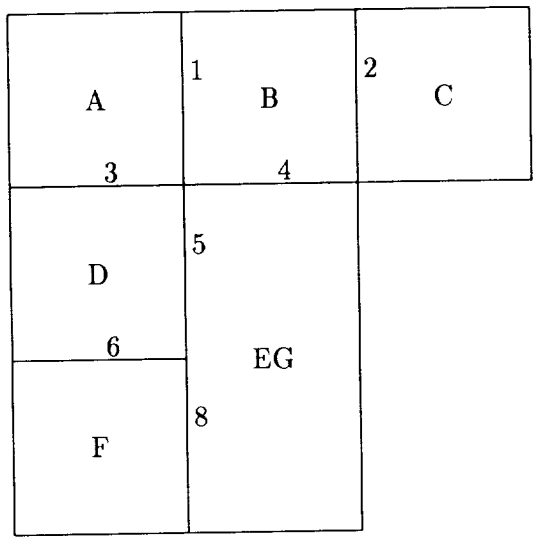



Figure 2: (Valid) configuration with edge 7 removed. Score=6.

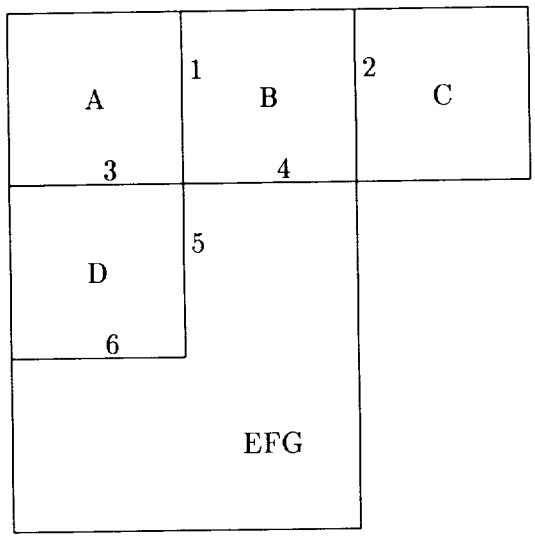



Figure 3: (Invalid) configuration with edges 7 and 8 removed.

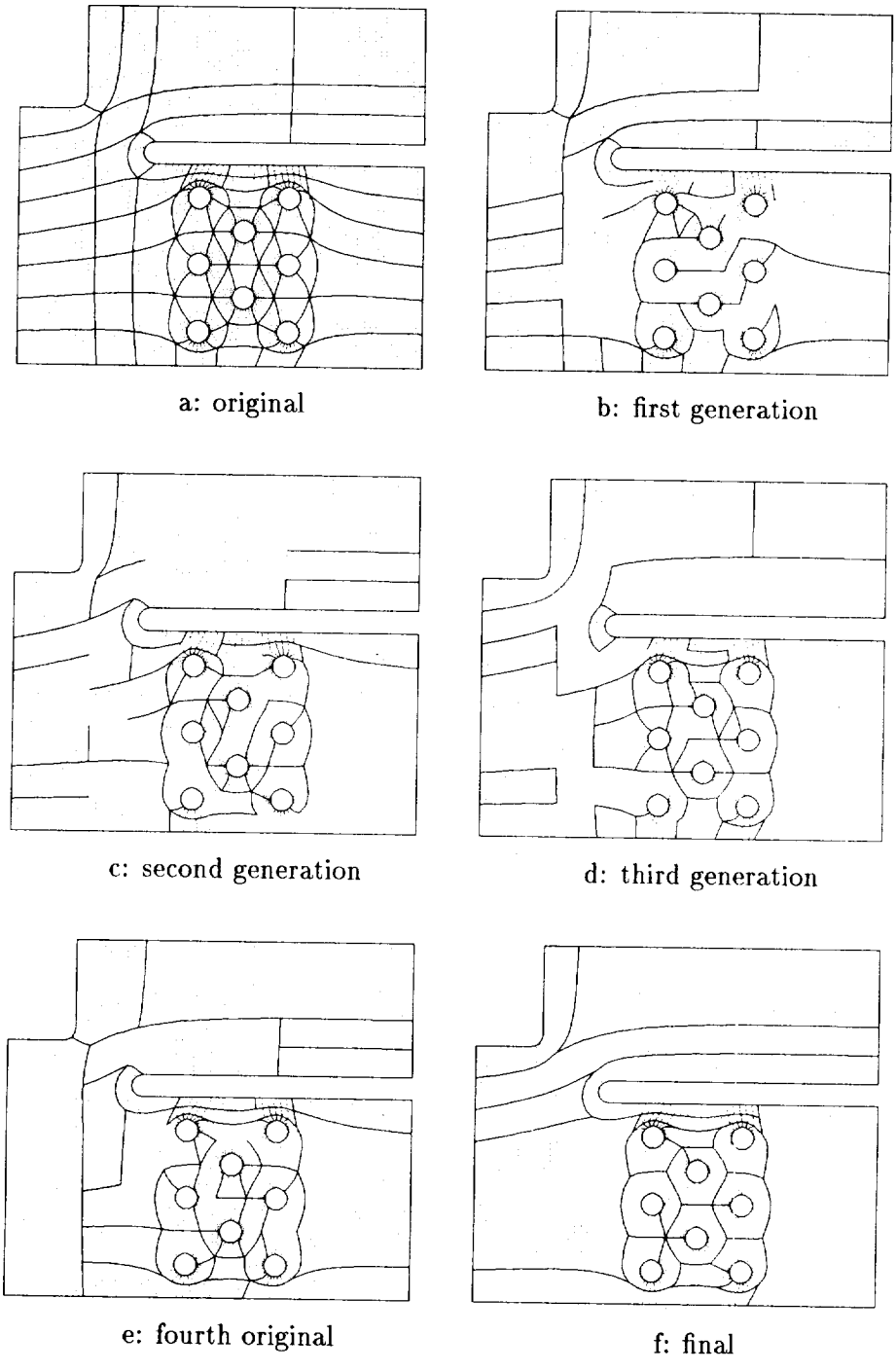



Figure 4: Minimization of the number of clusters for test case "posts".

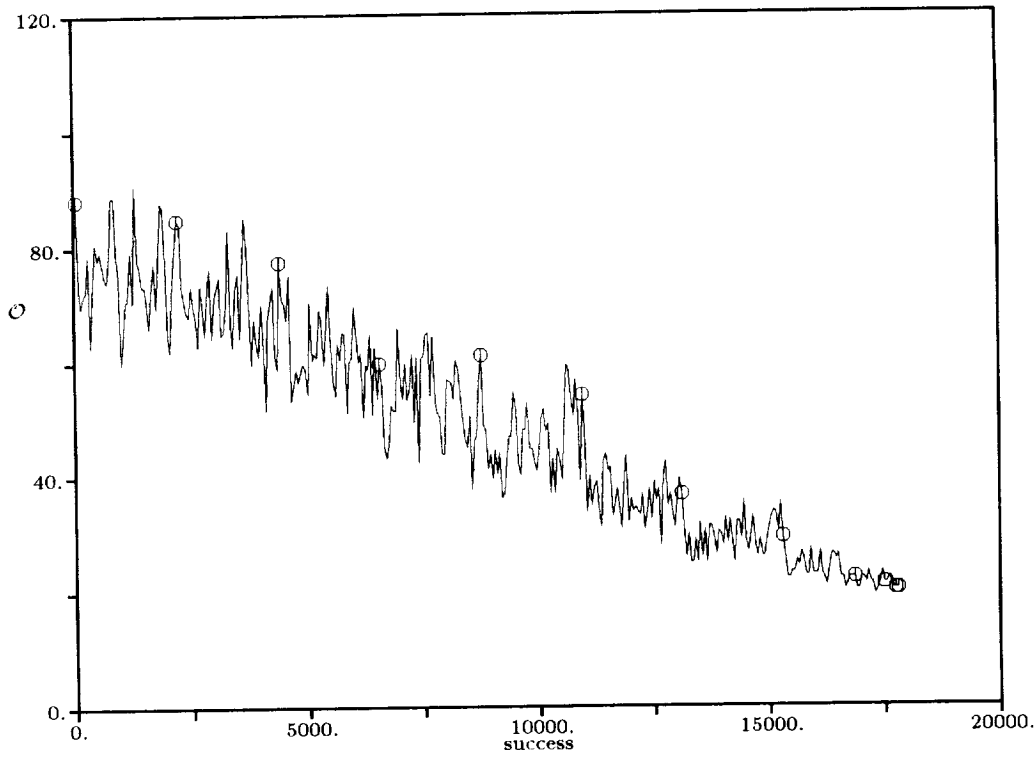
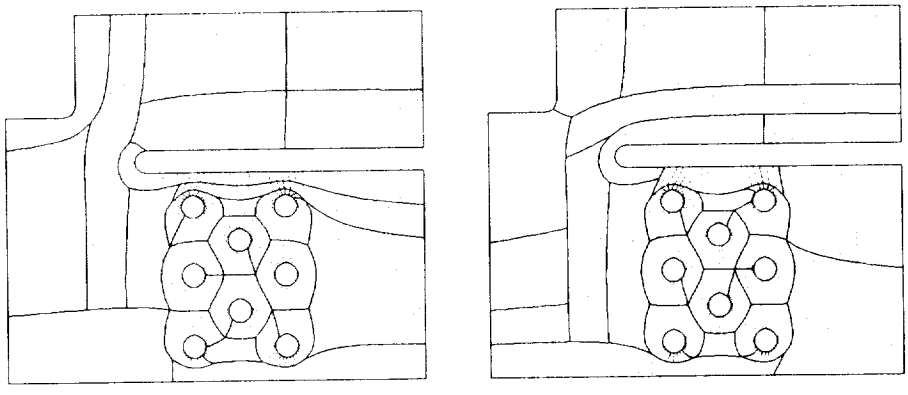
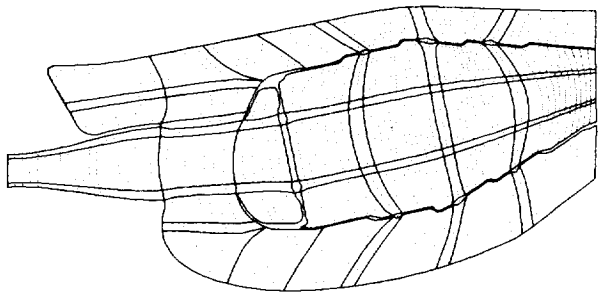



Figure 5: Convergence history for test case "posts". Only every 50th successful change is plotted. The circles show the times when the control parameter $T$ was decreased.

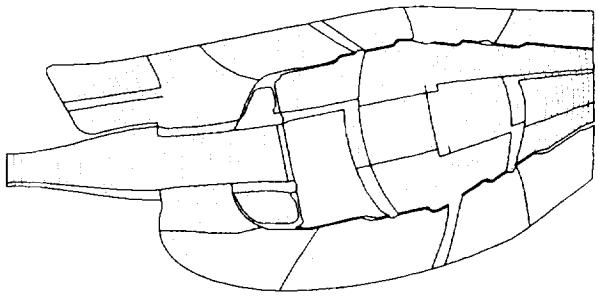


a: $L_{\text{size}} = 500$ b: $L_{\text{clus}} = 25$

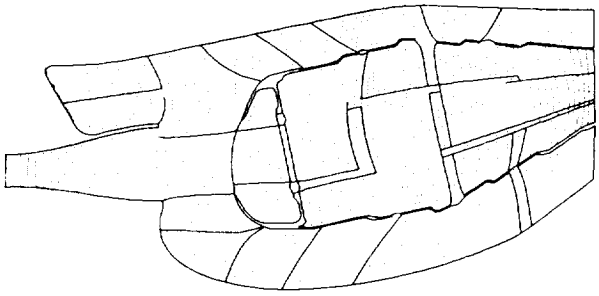
Figure 6: Final clusters for alternative optimizations for test case "posts".



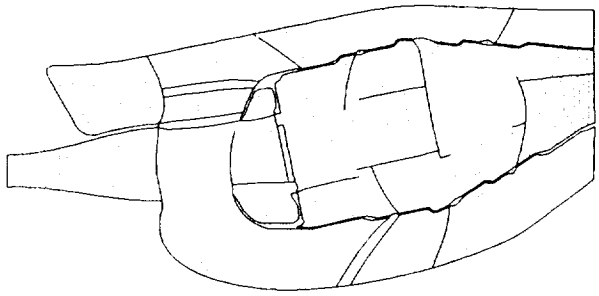
a: original



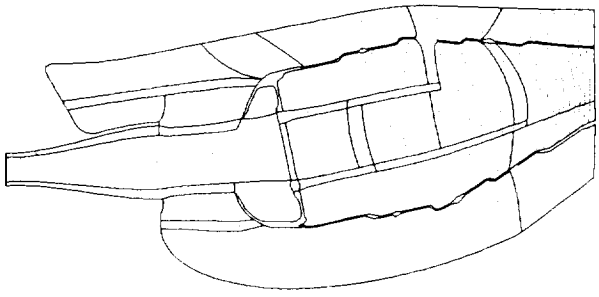
b: first generation



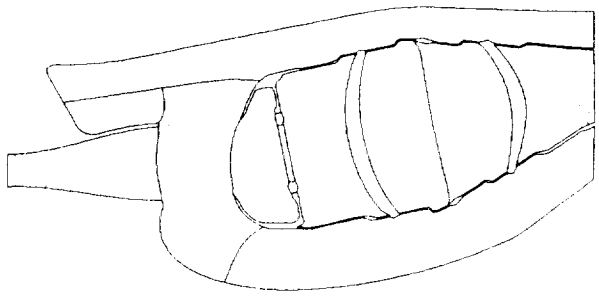
c: second generation



d: third generation

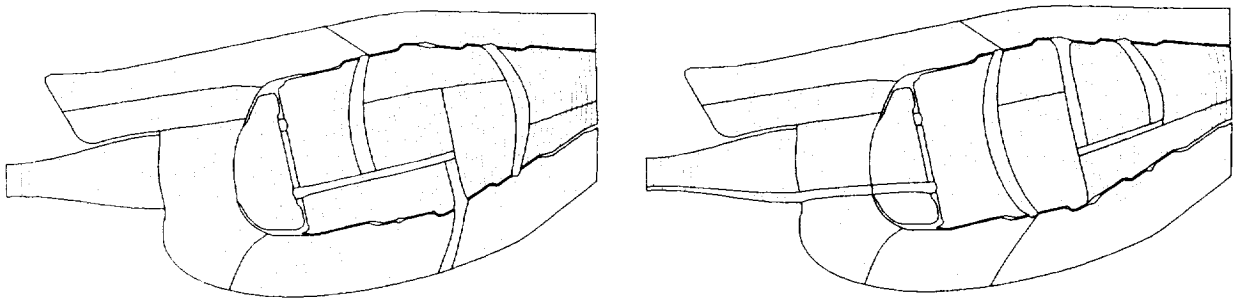


e: fourth original



f: final

Figure 7: Minimization of the number of clusters for test case "combustor".



a: $L_{\text{size}} = 500$ b: $L_{\text{clus}} = 25$

Figure 8: Final clusters for alternative optimizations for test case "combustor".

N95-28770

# A COMBINED GEOMETRIC APPROACH FOR COMPUTATIONAL FLUID DYNAMICS ON DYNAMIC GRIDS

John W. Slater [1]
NASA Lewis Research Center
Cleveland, Ohio 44135

## SUMMARY

A combined geometric approach for computational fluid dynamics is presented for the analysis of unsteady flow about mechanisms in which its components are in moderate relative motion. For a CFD analysis, the total dynamics problem involves the dynamics of the aspects of geometry modeling, grid generation, and flow modeling. The interrelationships between these three aspects allow for a more natural formulation of the problem and the sharing of information which can be advantageous to the computation of the dynamics. The approach is applied to planar geometries with the use of an efficient multi-block, structured grid generation method to compute unsteady, two-dimensional and axisymmetric flow. The applications presented include the computation of the unsteady, inviscid flow about a hinged-flap with flap deflections and a high-speed inlet with centerbody motion as part of the unstart / restart operation.

## INTRODUCTION

The computation of the unsteady fluid dynamics about mechanisms with components in relative motion has become an important topic in computational fluid dynamics (CFD) [1-5]. One example of such a mechanism is the NASA Variable Diameter Centerbody (VDC) inlet in which the axisymmetric centerbody can translate and change diameter to adjust the mass flow rate and stabilize the flow [6,7].

The CFD analysis process involves the aspects of geometry modeling, grid generation, and flow modeling, as shown in figure 1. When the components of the mechanism are not in relative motion, the geometry model, if one exists, is used to generate a grid, which is then used in the flow model and computation. The flow may, or may not, be a function of time. If no solution adaptive grid method is used, the grid remains fixed for the time interval of the analysis. This process is sequential with information being passed in one direction from the geometry model to the grid to the flow model.

When the components of the mechanisms are in relative motion, the geometry modeling, grid generation, and flow modeling all become a function of time. The main flow of information continues to be from the geometry model to the grid to the flow model. However, the geometry modeling and grid generation take on an increased importance. The interrelationships between the geometry modeling, grid generation, and flow modeling become more important and are discussed. The approach presented here attempts to use these interrelationships advantageously to enhance the overall computation of the total dynamics problem.

The focus of the approach is placed on the establishment of a geometry model, which is composed of a collection of geometric entities representing the components of the mechanism. This allows for accurate modeling of the mechanism and specification of the geometry motion. The existence of a geometry model allows the possibility for transfer of geometry data in the

[1]National Research Council Associate, CFD Branch, Internal Fluid Mechanics Division

form of standardized computer-aided-design (CAD) file formats such as the NASA-IGES format [8]. The work here considers planar geometry with the motion of a geometric entity assumed to be a rigid body motion.

The flow domain is the enclosed control volume in which the flow equations are solved. The modeling of the boundary of the domain is generally considered part of the geometry modeling because the domain boundary entities usually have the same mathematical form as the geometric entities. The approach presented here places a greater distinction between the geometric entities and the domain boundary entities because of the possibility of the relative motion of the geometric entities.

When there is relative motion of the components of the mechanism, it is possible that the grid will be regenerated in some manner at each time step of the computation. This requires an efficient grid generation approach. A multi-block, structured grid topology is used in the work presented here. This choice of topology is due to the desire for accurate and efficient computation of unsteady, viscous flows. It is assumed that the block topology remains fixed during the time interval of the computation. This restricts the extent of the geometry motion to moderate levels. The approach presented here involves the full use of the geometry and flow models in the grid generation. The grid generation information is associated with the geometric and domain boundary entities to ensure accurate generation of the boundary grid and grid dynamics. The association provides a natural formulation of the grid generation problem, which is used to implement some automation into the grid generation problem. An algebraic, sub-block method is used to efficiently generate the grid while providing for high grid quality. The same approach is used to compute the grid dynamics.

A mixed Eulerian-Lagrangian description of the integral form of the Navier-Stokes equations accounts for the motion of the geometry and flow domain. These equations are discretized for a cell-vertex, finite-volume approximation. The motion of the cell-faces of the finite-volume are based on the grid speeds. A time-accurate method is used to integrate the finite-volume equations. The approach presented here associates the flow boundary conditions with the geometric and domain boundary entities; allowing a natural specification of the boundary conditions and improving the association of the flow with the geometry model and grid.

The approach introduced above attempts to use the interrelationships between the aspects of geometry modeling, grid generation, and flow modeling for the full benefit of the computation of the total dynamics problem. Because of these interrelationships and the focus on the geometry modeling, the approach is referred to as the *combined geometric approach for CFD.*

The details of the approach are presented in the following sections. Some applications of the approach are presented in the analysis of a hinged flap with sinusoidal flap motion and the VDC inlet during the unstart / restart operation involving the motion of the centerbody.

## GEOMETRY MODELING

A mechanism is assumed to consist of a set of components defined by some assembly information. The components may move relative to each other according to some kinematic relation as a function of time. The configuration of the mechanism is the spatial relationship of the components at a certain time. The work presented here assumes that each component moves as a rigid body in translation and rotation about a point; and not as a deformable body.

A mechanism used to illustrate the concepts of the approach is the NASA Variable Diameter Centerbody (VDC) inlet [6,7]. The VDC inlet is a mixed-compression inlet being studied at the NASA Lewis Research Center as a concept for a high-speed inlet for transport aircraft. Figure

2 shows the mechanical design of the VDC inlet. The VDC inlet is designed to operate at a cruise Mach number of 2.5 with a 45% internal contraction of the flow using a biconic forward centerbody. The second and aft cones of the centerbody consist of overlapping leaves which form an umbrella mechanism which allows the diameter of the centerbody to change to vary the mass flow of air into the inlet. The centerbody may also translate to adjust the mass flow rate. There is a bleed slot in the centerbody and bleed holes on the forward interior of the cowl for the stabilization of the normal shock and improvement of the pressure recovery.

The geometry model of the mechanism consists of a collection of geometric entities which model the individual components. The geometric entities are defined numerically using a parametric coordinate related to the arclength along the entity. The methods of computer-aided-design (CAD) provide standard procedures for generating such models as mathematical curves and surfaces [9]. An effective numerical representation of the geometry is the non-uniform rational B-spline (NURBS) representation. The NASA-IGES format [8], a subset of the Initial Graphics Exchange Standard (IGES), provides a standard format for transfer of geometry data. In a design environment, one might expect a geometry model would already exist as part of the CAD effort and would be available for the CFD analysis.

The work presented here considers a planar geometry model as would be needed for a two-dimensional or axisymmetric CFD analysis. Each component of the mechanism is modeled as a single geometric entity ($m$), represented either as a line segment or a cubic spline curve using the parametric coordinate ($u$), which corresponds to the arclength along the geometric entity. Figure 3 shows the planar geometry model for the VDC inlet with the individual geometric entities identified.

Each geometric entity is assumed to move as a rigid body, which allows an efficient modeling of the geometry using the parametric coordinate since the mathematical description of the curves only needs to be computed once, at the start of the computation.

The variable centerbody of the VDC inlet is modeled by specifying the dynamics of certain entities. The change in diameter of the centerbody is modeled by the rotation of the geometric entities defining the second and aft cones about their respective points of rotation as shown in figure 3. The translation of the centerbody is modeled by the translation of the geometric entities defining the centerbody nose, second cone, aft cone, and aft centerbody. The dynamics of these geometric entities are defined by simple kinematic relations. Figure 4 shows a variation of the centerbody in which the centerbody has translated forward and the second-cone has rotated to match the angle of the nose cone.

Geometric information is obtained from the geometry model as a function of the geometric entity ($m$), parametric coordinate ($u$), and time ($t$). Information that is available is the position coordinates ($\vec{r}$), velocity ($\dot{\vec{r}}$), tangent vector ($\hat{t}$), normal vector ($\hat{n}$), acceleration ($\ddot{\vec{r}}$), second-derivative ($\vec{r}\,''$), and curvature ($\kappa$). A single subroutine in the code provides this information by referencing the geometry model and entity kinematic state for the given time.

This approach for the geometry modeling may seem excessive for a simple planar geometry, but it sets a framework for the handling a more complex geometry model for three-dimensional mechanisms.

## DOMAIN MODELING

The domain modeling involves defining the boundary of the flow domain. Part of the boundary will consist of the geometric entities of the geometry model. The remaining part of the boundary requires domain boundary entities to be defined such that the flow domain is enclosed.

The domain boundary entities are defined mathematically the same as the geometric entities of the geometry model. Thus, the domain boundary entities may be specified in the same manner as the geometry model. In figure 3, the domain boundary entities include the inflow, farfield, and outflow entities.

The relative motion of some of the geometric entities of the geometry model requires that some domain boundary entities be of variable geometric representation. These entities are called variable domain boundary entities. For the planar VDC inlet model shown in figure 3, the bleed slot entity will be of variable shape as the second cone and aft cone are rotated. As the centerbody is translated, the axis-of-symmetry and aft centerbody entities will change shape. Figure 4 shows the change of shape in these variable domain boundary entities.

Defining the flow domain is a secondary task in the CFD analysis and one suited for automation. Such automation will require the use of information about the geometry and flow models. The approach presented here provides a framework for providing that information.

## GRID GENERATION

When there is relative motion of the geometric entities, the grid becomes a function of time. The grid will then be regenerated in some manner for each time step in the CFD analysis. This requires an efficient grid generation method.

The work presented here uses a multi-block, structured grid topology with grid lines matching contiguously across blocks. The choice of this topology was due to the desire for accurate and efficient computation of viscous flows. Further, the problems of interest (VDC inlet) did not warrant a more general grid topology such as overset or unstructured grids. Yet, many of the ideas of the approach are applicable to other grid topologies.

For the planar domains of this work, a block is a quadrilateral. A face of the quadrilateral is defined by specifying the entities which comprise the face. To enclose the quadrilateral, block interface entities may have to be defined. The block interface entities also divide two blocks in the domain. These are represented in the same mathematical form as the geometric and domain boundary entities. Thus the block interface entities may be specified in the same manner as the geometry model. Figure 3 shows the block interface entity for the VDC inlet.

The block represents a $(\xi, \eta)$ generalized coordinate system. The generalized $\xi$-coordinate of the structured block is commonly directed in the streamwise direction while the generalized $\eta$-coordinate is commonly directed in the transverse flow direction. To reduce the amount of work required to generate a grid, each block is limited to only one entity in the $\eta$-direction ($\xi_{min}$ and $\xi_{max}$ faces).

It is assumed that the topology of the block remains fixed throughout the computation. This imposes a limitation on extent of the motion of the components and limits the generality of the approach, but it minimizes the amount of work required to regenerate the grid at each times step.

The grid generation is performed efficiently by the use of algebraic methods. Quality grids are obtained by sub-dividing a block into sub-blocks according to the geometric features of the entities. Each sub-block is usually small enough to approximate the shape of a generalized "rectangle". A face of a sub-block contains at most one entity and may contain only a portion of an entity.

Figure 5 shows how the domain for the VDC inlet has been divided into two blocks with sub-blocks. The first block extends from the inflow boundary through the diffuser to the compressor face. Block 2 extends from the inflow boundary over the cowl to the exterior outflow boundary.

Since there is only one entity along in the $\eta$-direction of a block, the entities that define the $\eta$-constant boundaries of each sub-block are defined. The $\xi$-constant boundaries of the sub-block are either the $\xi_{min}$ boundary of the block, the $\xi_{max}$ boundary of the block, or a sub-block boundary. The sub-block boundary is defined using a two-point cubic spline with the endpoint tangents specified as the normal vectors of the entities on the $\eta_{min}$ and $\eta_{max}$ faces of the sub-block. Figure 5 shows how these sub-block boundary curves intersect the entities at right angles. This orthogonality helps generate an interior grid that is orthogonal to the entities.

The endpoint normal vector used to construct the sub-block boundary curve may have different directions if the slopes of the entities at a junction of two entities do not match. This is illustrated in figure 6. The choice of which normal vector to use is based on the flow boundary conditions for the entities. For example, if entity B is a solid wall boundary condition while entity A is a freestream boundary condition, the normal vector from entity B would be used since it is felt that it is more important to resolve the solid wall flow than the freestream flow.

One feature of the combined geometric approach is that the grid generation information such as grid density, grid spacing along the boundary, and grid spacing normal to the boundary is associated with the geometric entities. This allows for a more natural procedure for the specification of this information. One can sense the number of grid points and the spacings that may be needed to resolve the flow along and normal to an entity. One constraint is that the number of grid points need to match across the block and not introduced excessive grid line skewing.

Once, the geometry of the sub-block is defined, the grid density and spacing are defined along the boundaries. A hyperbolic tangent method is generally used to distribute the boundary grid points with respect to the parametric coordinate of the geometric entity [10]. Each boundary grid point is associated with a parametric coordinate on a geometric entity. This ensures that the boundary grid will always adhere to the time-dependent geometry.

The association of the grid and geometry allow geometric information such as position coordinates ($\vec{r}$), velocity ($\dot{\vec{r}}$), tangent vector ($\hat{t}$), normal vector ($\hat{n}$), acceleration ($\ddot{\vec{r}}$), second-derivative ($\vec{r}\,''$), and curvature ($\kappa$) to be available for use in the grid generation methods.

In a CFD analysis, grid generation is a secondary activity in that it is a means to an end: the computation of the flowfield. Ideally, grid generation should be fairly automatic and transparent to the analyst. Proper automation of grid generation requires information on the geometry and flow conditions. The interrelationships between the geometry model, grid, and flow model as highlighted in this approach lends itself to an automated procedure for determining the grid density and spacing along the entities. Using information on the geometric properties of the entity, the flow boundary conditions, and grid quality parameters such as minimum and maximum grid spacing, maximum grid spacing ratio, and minimum and maximum grid aspect ratio, the appropriate number of grid points and their spacing are determined along each entity.

Once the grid points are distributed along the sub-block boundaries, the volume grid can be generated using a transfinite-interpolation method. Since a sub-block maintains a fairly "rectangular" shape, a transfinite-interpolation works well.

Figure 7, 8, 9 show examples of grid generated using the sub-block, algebraic grid method. The grid for the VDC inlet is shown for the entire domain in figure 7, while figure 8 shows a close-up of the grid in the throat region of the VDC inlet with the centerbody moved. Figure 9 shows the grid for the hinged-flap at a 15 degree angle-of-attack.

The motion of the geometry requires the grid motion to be computed for each time step. Several strategies are investigated for generating the dynamic grid and computing the grid speeds. First, one can regenerate the grid at each point in time based on the current configuration

of the geometric model. The grid speeds can then be determined through a time-difference of a sequence of grids. A second strategy is to start with an initial grid and deform it as a geometric entity is moved. Again the grid speeds can be determined through a time-difference. A third strategy is to start with an initial grid and directly compute the grid speeds as a boundary-value-problem using the knowledge of the velocities of the entities. The time-dependent grid is then obtained through a time-integration of the grid speeds.

## FLOW MODELING

The integral form of the Navier-Stokes equations for a time-varying control volume is

$$\frac{d}{dt}\int_{V(t)} U\, dV + \oint_{S(t)} \mathbf{H}\cdot\hat{n}dS = 0 \tag{1}$$

where $U$ is the algebraic vector of conservative variables,

$$U^T = \left(\rho,\ \rho\vec{V},\ E_t\right) \tag{2}$$

where for a perfect gas,

$$E_t = \rho\left[e + \frac{1}{2}\left(\vec{V}\cdot\vec{V}\right)\right]$$

and $p$ and $\rho$ are the primitive variables of pressure, and density. The velocity vector, $\vec{V}$, in Cartesian coordinates is

$$\vec{V} = u\,\hat{\imath} + v\,\hat{\jmath}. \tag{3}$$

The $\mathbf{H}$ is the flux dyadic, which for a mixed Lagrangian-Eulerian description [11] is,

$$\mathbf{H} = \mathbf{F} - \vec{g}\,U. \tag{4}$$

The $\vec{g}$ is the velocity vector of the boundary of the control volume. In Cartesian coordinates, it is

$$\vec{g} = x_\tau\,\hat{\imath} + y_\tau\,\hat{\jmath}. \tag{5}$$

An Eulerian description is obtained for $\vec{g} = 0$ while a Langrangian description is obtained for $\vec{g} = \vec{V}$.

The $\mathbf{F}$ is the Cartesian flux dyadic for the two-dimensional, unsteady Navier-Stokes equations. The flow equations are complete with Sutherland's formula, the definition of the Prandtl number, and the assumptions of a perfect gas (air) and laminar viscous flow. A Reynolds averaging is used to approximate turbulence effects and a turbulence model provides the eddy viscosity.

The flux dyadic can also be expressed as

$$\mathbf{H} = \mathbf{C} - \mathbf{D}, \tag{6}$$

where $\mathbf{C}$ is the portion of the flux dyadic containing the convective terms,

$$\mathbf{C} = \left(\vec{V} - \vec{g}\right)U, \tag{7}$$

and $\mathbf{D}$ is the portion of the flux dyadic containing the non-convective terms.

Equation 1 can be expressed in a compact form of

$$\frac{d\hat{U}}{dt} + \hat{R} = 0 \tag{8}$$

where

$$\hat{U} = \int_{V(t)} U \, dV \tag{9}$$

and

$$\hat{R} = \oint_{S(t)} \mathbf{H} \cdot \hat{n} dS. \tag{10}$$

The solution of the flow equations require the specification of the flow boundary conditions. One feature of the combined geometric approach is that the boundary conditions are associated with the entities of the geometry model. For example the curves defining the surfaces of the inlet (i.e. nose cone, cowl, etc...) are specified to be solid wall boundary conditions which are slip surfaces for inviscid flow analysis and non-slip walls for viscous flow analysis. The flow domain boundaries not representing the inlet surfaces are specified to be inflow or outflow boundaries. Associating the flow boundary conditions with entities allows the possibility that for a moving geometry and grid, the boundary grid points move from one geometric entity to another. Thus, the grid point may change the type of boundary condition with which it is associated.

One issue that arises is what boundary condition to impose at a grid point located at the junction of two entities if the entities are associated with different boundary condition types? These issues are resolved by defining a hierarchy for the flow boundary conditions. For example, at the junction of an entity defined as a non-slip wall and an entity defined as an inflow boundary, the non-slip wall would get more priority and the boundary grid point would be imposed as a non-slip boundary condition.

The existence of a geometry model allows its use by the flow boundary condition modeling. The physical boundary condition for a slip wall can be specified as

$$\rho_B \left( \vec{V}_B - \vec{g}_B \right) \cdot \hat{n} = \dot{m}_B. \tag{11}$$

Commonly, $\hat{n}$ is computed from the local grid. Since the boundary grid point is associated to a geometric entity through a parametric coordinate, the exact normal vector can be determined easily. Further, the pressure at the surface can be determined through a normal projection of the of the momentum equation at the wall,

$$\frac{\partial p}{\partial n} = \kappa \, \rho_B \left[ \left( \vec{V}_B - \vec{g}_B \right) \cdot \hat{t} \right]^2. \tag{12}$$

Again the geometry model can be accessed to obtain the tangent vector $\hat{t}$ and the curvature of the wall $\kappa$.

## TIME-DEPENDENT COMPUTATION OF THE DYNAMICS

A cell-vertex, finite-volume approximation is applied for the spatial discretization of equation 8. The discretization allows for two-dimensional and axisymmetric planar flow domains. The temporal discretization is applied using an explicit, two-stage Lax-Wendroff method [12]. The time-marching algorithm for a time step $\Delta\tau$ is of the form

$$\hat{U}_{i,j}^{*} = \hat{U}_{i,j}^{n} + \Delta\tau \, \hat{R}_{i,j}^{n}, \tag{13}$$

$$\hat{U}_{i,j}^{**} = \hat{U}_{i,j}^{*} + \Delta\tau\ \hat{R}_{i,j}^{*}, \tag{14}$$

and

$$\hat{U}_{i,j}^{n+1} = \frac{1}{2}\left\{\ \hat{U}_{i,j}^{n} + \hat{U}_{i,j}^{**}\ \right\}. \tag{15}$$

The $\hat{U}$ and $\hat{R}$ denote the algebraic vectors of the generalized conservative variables and flux residual, respectively. The $\hat{U}$ is defined as

$$\hat{U} = U\ V, \tag{16}$$

where $V$ is the volume of the finite-volume cell. The $\hat{R}$ is defined as

$$\hat{R}_{i,j} = \hat{F}_{i+1/2,j} - \hat{F}_{i-1/2,j} + \hat{F}_{i,j+1/2} - \hat{F}_{i,j-1/2}, \tag{17}$$

where the flux vector for a cell face is defined as

$$\hat{F} = \mathbf{H}\cdot\hat{n}dS. \tag{18}$$

The inviscid fluxes are computed using the Roe flux-difference splitting with a TVD limiter as presented in reference [12]. The viscous fluxes are computed using differences and averages computed at the cell faces. The time step is computed using the CFL condition.

Characteristic boundary conditions are used to compute the solution points on the boundaries. The flow conditions at the compressor face boundary of the VDC inlet were applied using a variation of the method discussed in reference [13]. This involved specifying the average Mach number at the compressor face, which allowed the pressure to vary and was shown to be a non-reflecting boundary condition.

The $V$ needed to decode $\hat{U}$ is computed from the geometric conservation law as discussed in references [14] and [15]. The geometric conservation law relates the change in volume of the cell to the motion of the cell faces and is derived from the flow integration equations with the assumption of a uniform solution for the conservative variables. The geometric conservation law follows the form of equations (13) to (15) with $\hat{U} = V$ and $\hat{R} = \hat{Z}$ where $\hat{Z}$ is the vector sum of speeds of the cell faces.

## APPLICATIONS

Some preliminary inviscid computations involving dynamic grids are presented below.

### Simple Hinged Flap

A simple mechanism with relative motion of its components is a plate with a hinged flap at the end of the plate. The flap is specified to rotate about its hinge in a sinusoidal manner with an amplitude of 15 degrees and a period of 0.03 seconds. The time scale of the motion is perhaps unrealistic for a flap or aileron motion on an aircraft; however, it serves to demonstrate the dynamic grid capabilities.

The analysis assumes an inviscid flow with a freestream Mach number of 2.0. The steady-state flow is obtained with the flap at zero degrees deflection and no movement of the geometry. Once the steady-state flow has been obtained, the flap is set in motion. Figure 9 shows the grid with a flap deflection of 15.0 degrees. The time-variation of the Mach number contours is shown in figure 10. The time-varying formation and disappearance of the shocks and expansions can be seen. To get some evaluation of the accuracy of the flow computation, the variation of the Mach

number at the midpoint of the upper surface of the flap during the first and third periods of the flap motion was compared to the Mach number computed from steady flow, inviscid supersonic theory [16]. Figure 11 shows that the comparison is fairly good. The computed Mach number for the compression portion of the flap motion compares well to the steady flow theory ; however, in the expansion portion, the Mach number compares less. This suggests that dynamic effects may be more significant in the expansion region.

The flap motion problem is similar to other problems of interest in aeropropulsion. One such problem is the unsteady flow about a thrust-vectored nozzle.

## NASA VDC Inlet

The unsteady flow through the VDC inlet during the unstart / restart operation is now analyzed. At the cruise conditions, the inlet is in a supercritical mode in which a normal shock is positioned just aft of the centerbody bleed slot. This allows for a shorter diffuser, which saves on weight of the inlet, and for a maximum total pressure recovery. The disadvantage of this shock placement is that the shock is sensitive to flow perturbations and may unstart. During unstart, the shock moves forward in the inlet. Once past the throat, the shock is forced to be expelled from the inlet. This creates a bow shock ahead of the cowl lip, resulting in significant drag. A buzz cycle will develop if the inlet is not restarted. For the VDC inlet, restart is achieved through a slight forward translation of the centerbody, a reduction in the diameter of the centerbody, and an opening of the bypass doors to reduced the back pressure at the compressor face.

A preliminary inviscid flow computation is presented to show the nature of the flowfield during the unstart / restart operation. The steady-state flow is computed for a freestream Mach number of 2.5 with the inlet at an angle-of-attack of 0.0 degrees. The Mach number at the compressor face was set at 0.31, which is just slightly greater than the design compressor face Mach number. This places the terminal shock at a position $x/r_{cowl} = 3.25$, which is greater than the design position of $x/r_{cowl} = 2.73$.

The computed pressures along the forward portion of the centerbody compared very well with data from wind tunnel tests [7]. However, comparisons in the diffuser were not comparable since the actual flow is dominantly viscous.

The inlet is forced to unstart through an impulse at the compressor face of the Mach number of a magnitude of -0.12 over a time interval of 0.01 seconds. The pressure pulse that is created at the compressor face travels upstream in the diffuser and interacts with the terminal shock, which is then forced forward. The shock passes ahead of the throat. When the shock reaches the shock sensor, the restart process begins with a start in the motion of the centerbody. For this analysis, the centerbody was set to translate a distance of 0.2 units forward and the second cone angle rotates from an angle of 18.5 degrees to a angle of 12.5 degrees, which is equal to the angle of the nose cone. The centerbody motion occurs over a time interval of 0.05 seconds. The compressor face Mach number is increased from a value of Mach 0.31 to 0.50 to simulate the opening of the bypass doors and the reduction of the back pressure. Figure 12 shows a sequence of Mach number contours for the unstart / restart operation. The shock is expelled from the inlet and proceeds a short distance ahead of the cowl lip before returning to the cowl lip and entering the inlet. A new terminal shock begins to form in the diffuser as the flow ahead of it becomes supersonic. Once the terminal shock is formed, the centerbody returns to its design position and the compressor face Mach number returns to a value of 0.31.

The analysis of the unstart / restart flowfield continues. At this point it is evident that the motion of the centerbody is significant to the restart operation.

## CONCLUSIONS

For the CFD analysis of a mechanism involving the relative motion of its components, the aspects of geometry modeling and grid generation become a function of time, as does the flow modeling, and take on an increase importance in the problem. The combined geometric approach presented in this paper has examined some of the interrelationships between the geometry modeling, grid generation, and flow modeling and has shown that these interrelationships provide for a more natural formation of the problem and can be used advantageously in the computation of the total dynamics problem.

## ACKNOWLEDGEMENT

We would like to thank John D. Saunders of the NASA Lewis Research Center for providing geometry and wind tunnel test data for the VDC inlet.

## REFERENCES

1. Mani, K.K. and J.W. Haney, "3D CFD Analysis of a SR71-Waverider Launch Configuration," AIAA-94-0156, January 1994.

2. Atwood, C.A., "Computation of a Controlled Store Separation from a Cavity," AIAA-94-0031, January 1994.

3. Wang, Z.J. and H.Q. Yang, "Unsteady Flow Simulation Using a Zonal Multi-Grid Approach with Moving Boundaries," AIAA-94-0057, January 1994.

4. Trépanier, J.Y., M. Reggio, M. Paraschivoiu, and R. Camarero, "Unsteady Euler Solutions for Arbitrarily Moving Bodies and Boundaries," AIAA Journal, Vol. 31, No. 10, p. 1869-1876.

5. Meakin, R.L., "Moving Body Overset Grid Methods for Complete Aircraft Tiltrotor Simulations," AIAA-93-3350, July 1993.

6. Wasserbauer, J.F., R.J. Shaw, and H.E. Neumann, "Design of a Very-Low-Bleed Mach 2.5 Mixed-Compression Inlet with 45 Percent Internal Contraction," NASA TM X-3135, 1975.

7. Saunders, J.D. and A.A. Linne, "Status of the Variable Diameter Centerbody Inlet Program," First Annual High-Speed Research Workshop. pp 1481-1504, NASA CP-10087, Part 3, April 1992. (Limited distribution).

8. Blake, M.W., P.A. Kerr, S.A. Thorp, J.J. Chou, "NASA Geometry Data Exchange Specification for Computational Fluid Dynamics (NASA IGES)," NASA RP 1338, April 1994.

9. Zeid, Ibrahim, CAD/CAM Theory and Practice, McGraw-Hill, Inc., New York, 1991.

10. Anderson, D.A., J.C. Tannehill, and R.H. Pletcher. Computational Fluid Mechanics and Heat Transfer. New York, McGraw-Hill Book Company, 1984.

11. Vinokur, M. "An Analysis of Finite-Difference and Finite-Volume Formulations of Conservation Laws." Journal of Computational Physics. 81 (1989): 1-52.

12. Liou, M.-S. and A.T. Hsu. "A Time-Accurate Finite Volume High Resolution Scheme for Three Dimensional Navier-Stokes Equations." AIAA Paper 89-1994, June 1989.

13. Chung, J., "Numerical Simulation of a Mixed-Compression Supersonic Inlet Flow," AIAA Paper 94-0583, January 1994.

14. Thomas, P.D. and C.K. Lombard. "Geometric Conservation Law and Its Application to Flow Computations on Moving Grids." AIAA Journal 17 (1979): 1030-1037.

15. Hindman, R.G. "Generalized Coordinate Forms of Governing Fluid Equations and Associated Geometrically Induced Errors." AIAA Journal 20 (1982): 1359-1367.

16. Anderson, J.D., Modern Compressible Flow. New York, McGraw-Hill Book Company, 1982.

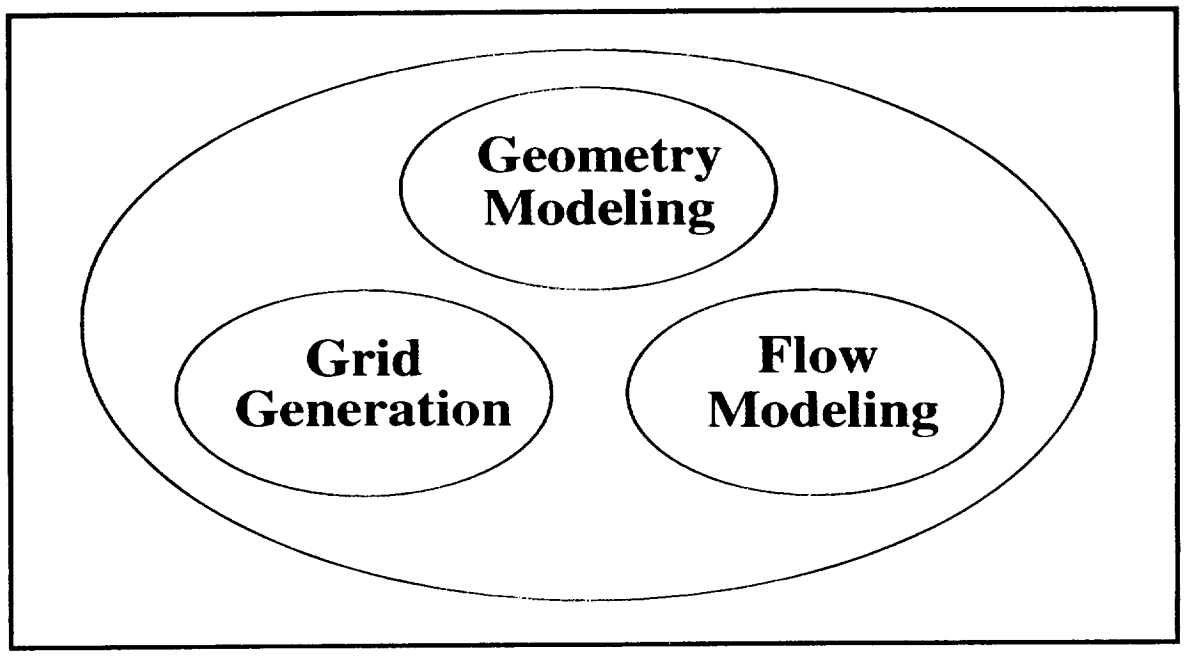



Figure 1: The aspects of the CFD analysis process: geometry modeling, grid generation, and flow modeling

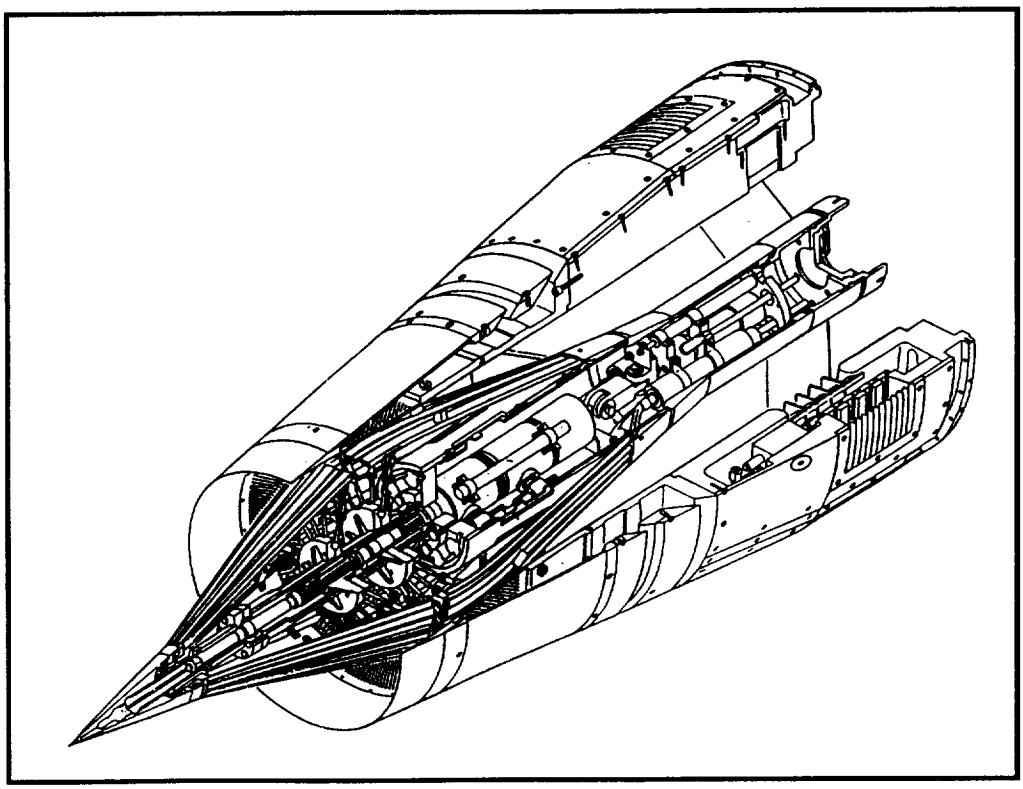


Figure 2: The mechanical design of the NASA Variable Diameter Centerbody (VDC) inlet

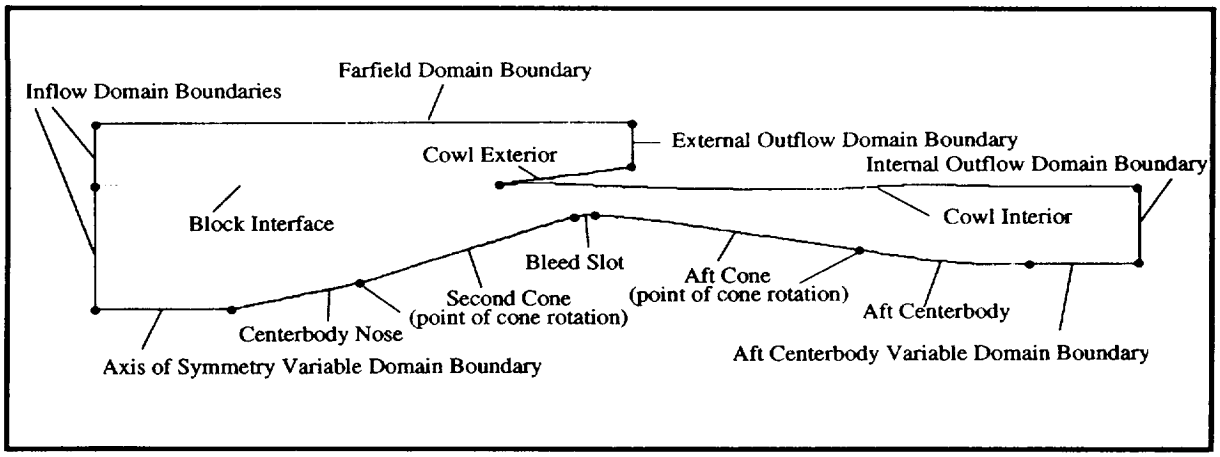



Figure 3: The entities of the planar geometry, flow domain, and block interface for the VDC inlet

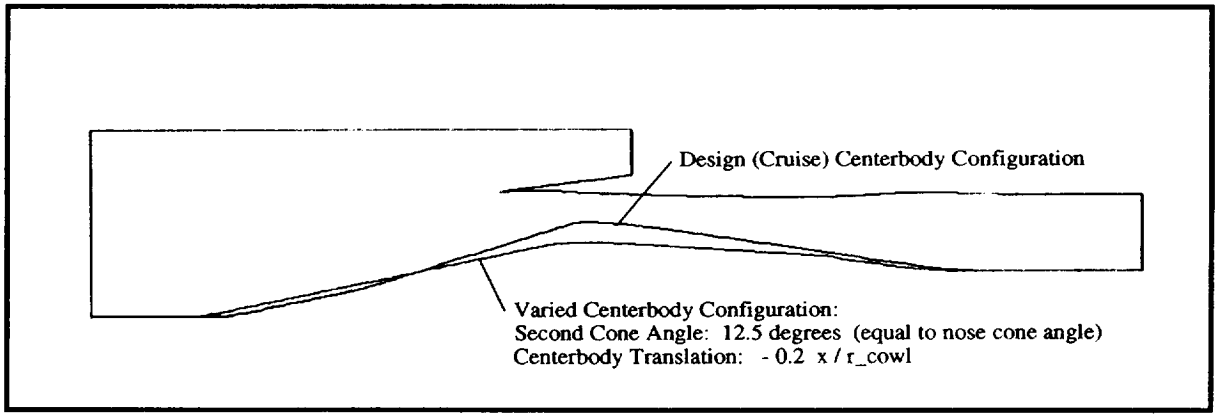



Figure 4: An example of the geometric variation of the VDC inlet

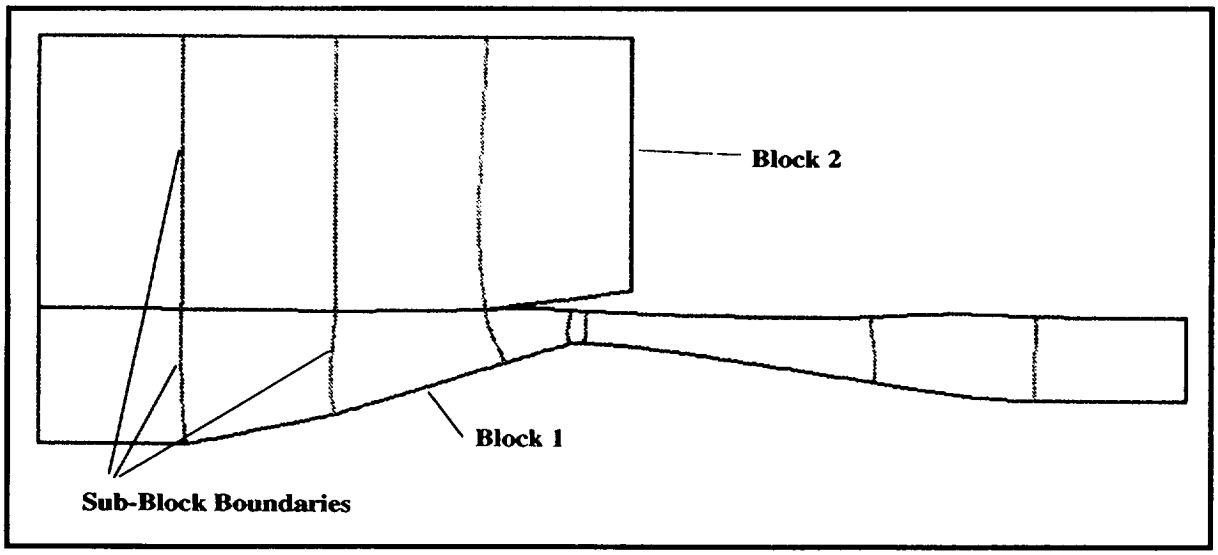



Figure 5: The block and sub-block boundaries for the grid for the VDC inlet

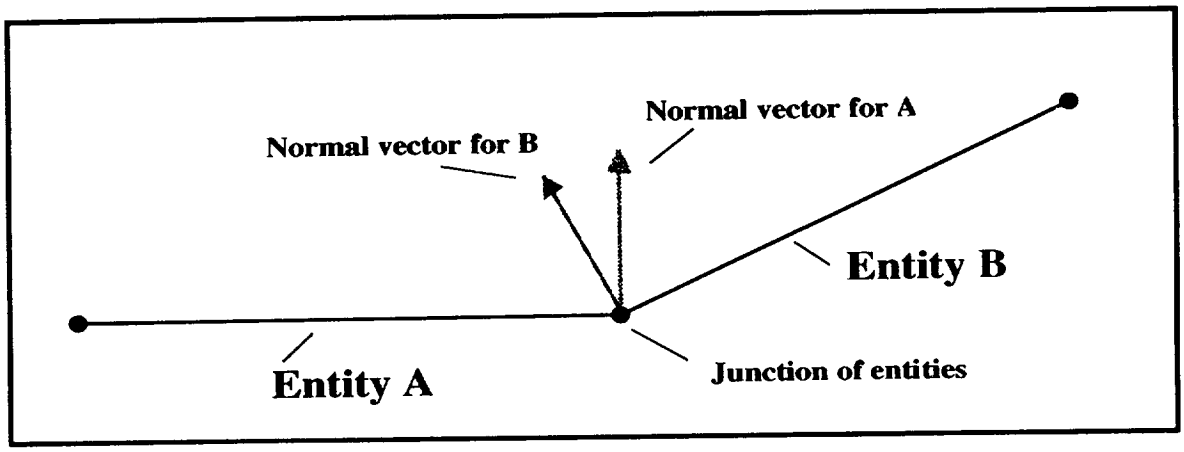



Figure 6: The junction of two entities of a planar geometry

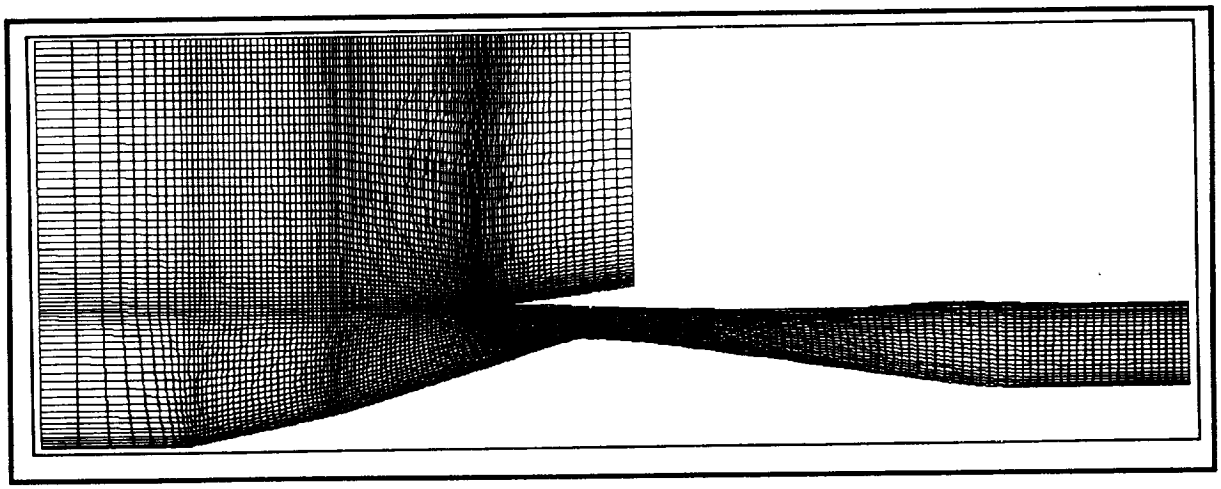


Figure 7: The multi-block grid for the inviscid analysis of the VDC inlet during the unstart / restart operation

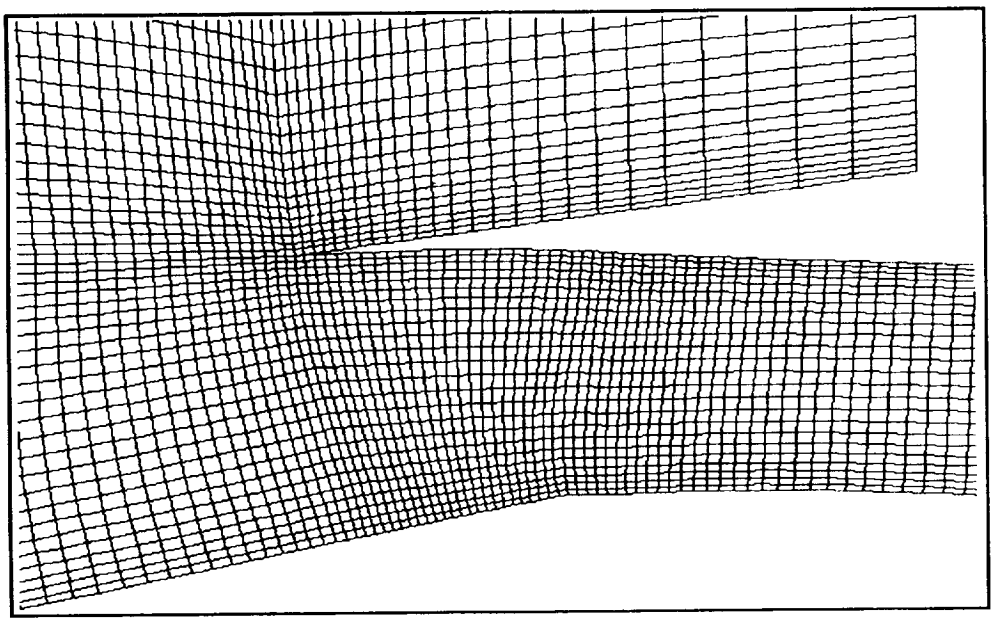


Figure 8: A closeup of the grid in the throat region of the VDC inlet with the centerbody translated and second and aft cones rotated

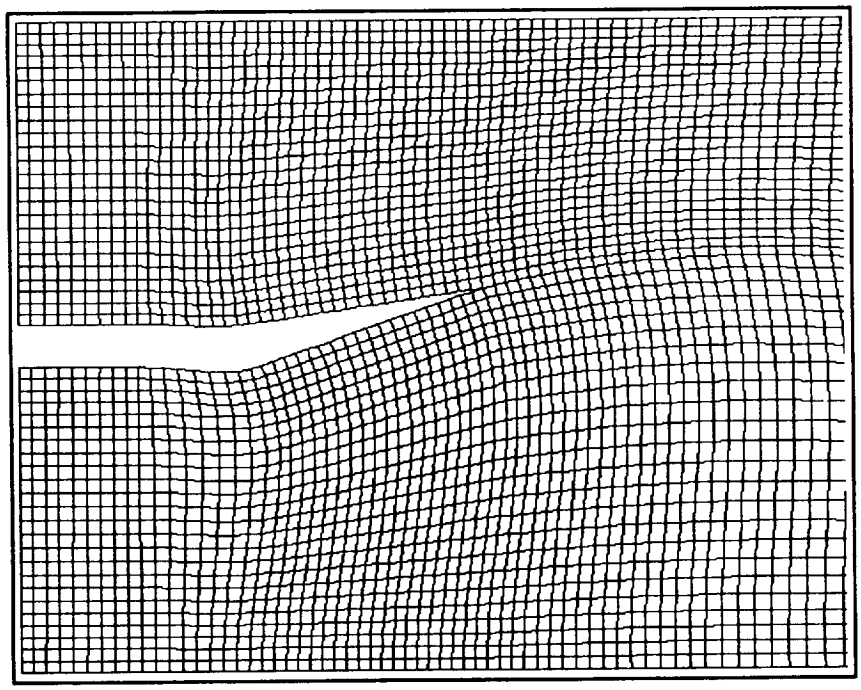


Figure 9: The grid for the hinged flap at a hinge angle of 15.0 degrees

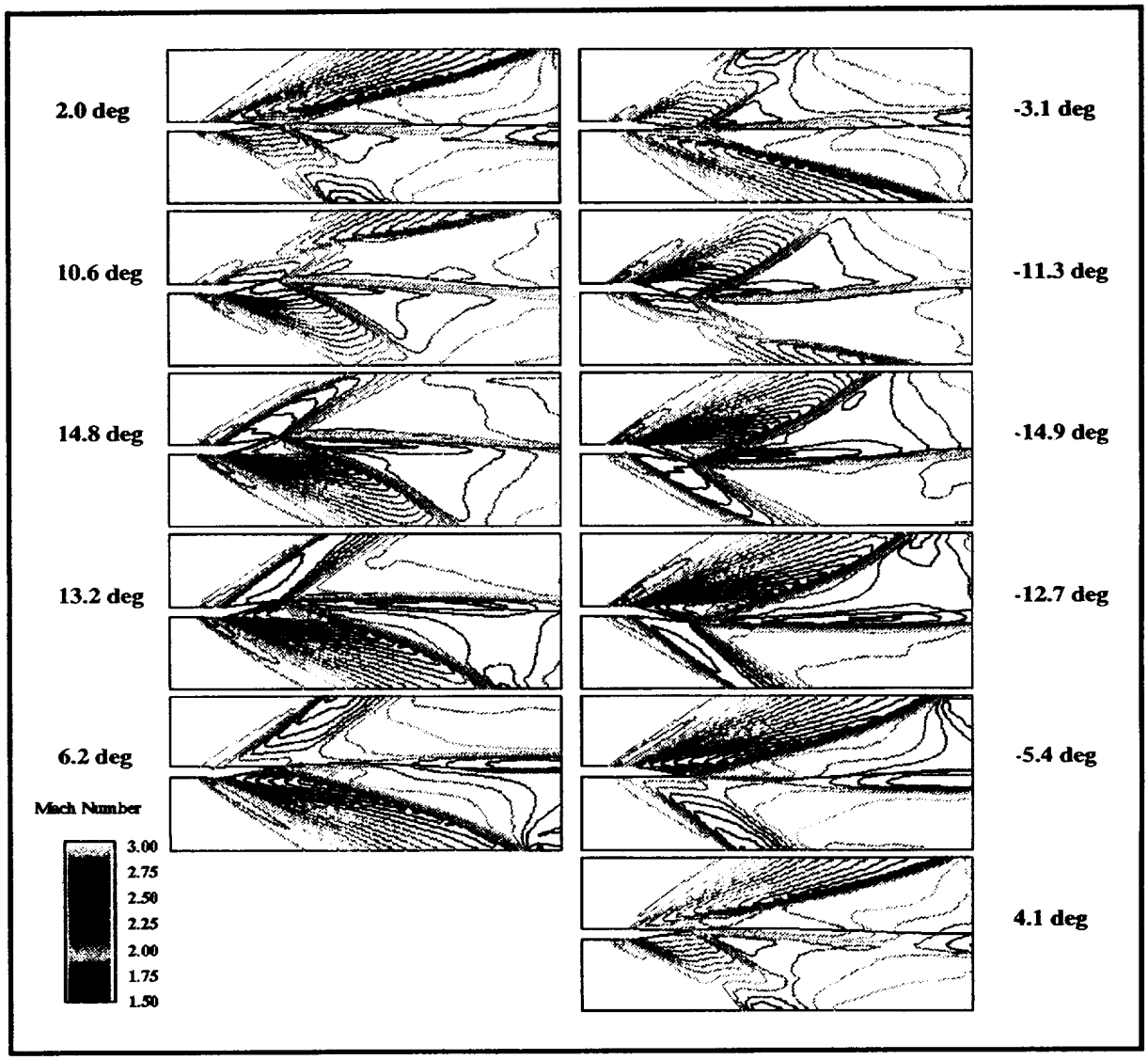



Figure 10: The sequence of Mach number contours for the inviscid analysis of the hinged flap with a freestream Mach number of 2.0 and zero degrees angle-of-attack

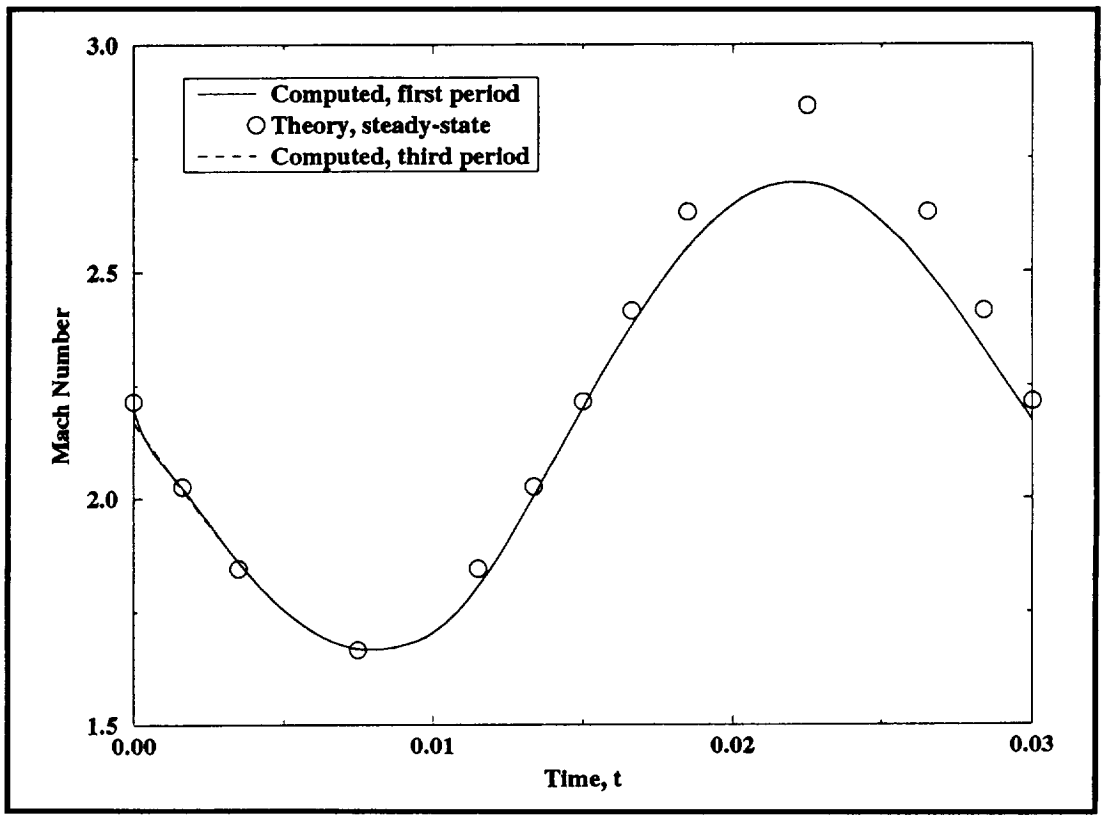



Figure 11: The comparison of the time-variation of the computed Mach numbers with the steady-state, oblique shock theory at the midpoint of the top surface of the hinged flap

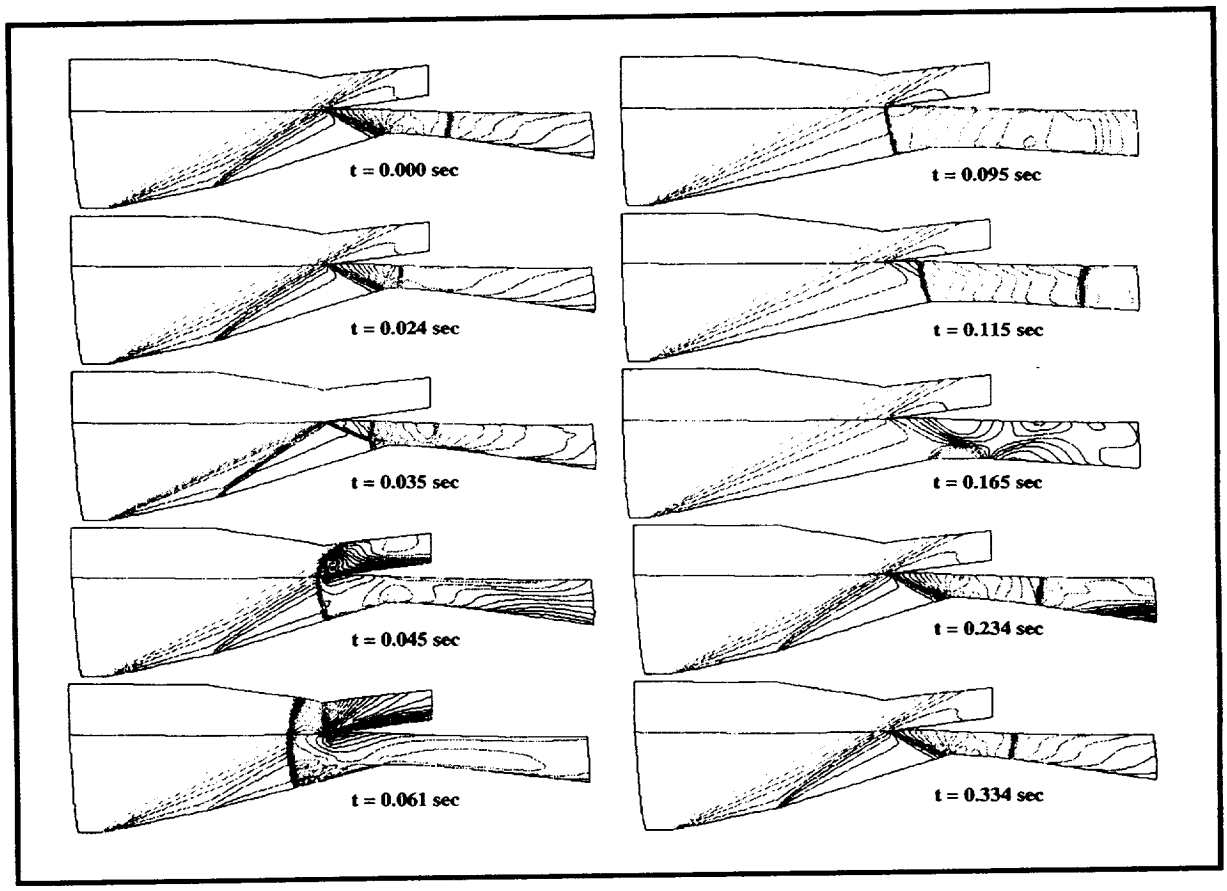



Figure 12: The sequence of Mach number contours for the inviscid analysis of the unstart / restart operation of the VDC inlet with centerbody translation and change of diameter

# SOFTWARE SYSTEMS (2)

N95- 28771

# PARAMETRIC DESIGN AND GRIDDING THROUGH RELATIONAL GEOMETRY

John S. Letcher, Jr. and D. Michael Shook
AeroHydro, Inc.
P. O. Box 684
Southwest Harbor, Maine 04679

## SUMMARY

Relational Geometric Synthesis (RGS) is a new logical framework for building up precise definitions of complex geometric models from points, curves, surfaces and solids. RGS achieves unprecedented design flexibility by supporting a rich variety of useful curve and surface entities. During the design process, many qualitative and quantitative relationships between elementary objects may be captured and retained in a data structure equivalent to a directed graph, such that they can be utilized for automatically updating the complete model geometry following changes in the shape or location of an underlying object. Capture of relationships enables many new possibilities for parametric variations and optimization. Examples are given of panelization applications for submarines, sailing yachts, offshore structures, and propellers.

## INTRODUCTION

Relational Geometric Synthesis (RGS) [1, 2] is a novel approach to 3-D geometric design, with many advantages over previous technology:

(1) RGS provides a new, transparent, object-oriented conceptual framework for building up complex models from a logical hierarchy of point, curve, surface and solid entities.

(2) A wide variety of useful point, curve, surface and solid entities are supported in a unified way, so that they can be used essentially interchangeably in the construction of other objects.

(3) During the design sequence, qualitative and quantitative relationships between objects are captured in such a way that they are automatically enforced during updates of the model, following changes in underlying objects.

(4) To a great degree, RGS avoids the common difficulties of surface-surface intersections and trimming that plague conventional B-rep solid modeling, by providing convenient ways to create surfaces that join accurately in the first place, and that remain accurately joined through capture and enforcement of this qualitative relationship.

(5) RGS models can be stored in an extremely compact format conveying only the "logical model", i.e. the "genetic instructions" for recreating the model from scratch, including all defined relationships between objects.

RGS clearly has some features in common with Armit's TAG [3] and Boeing's AGPS [4, 5, 6]. The extent of similarities to these systems is difficult to judge from the available references, but we believe RGS provides much more pervasive use of dependency relationships than AGPS, and a much richer entity set than either TAG or AGPS. Automatic update of geometry following changes is also characteristic of "constraint-based" modeling systems [7]. In our opinion, RGS is qualitatively different



PRECEDING PAGE BLANK NOT FILMED

from constraint-based modeling and in particular is much more adept at supporting 3-D design with free-form curves and surfaces.

Compared with NURBS-based curve and surface modeling, RGS's explicit support of many entities not precisely representable by NURBS has large advantages in modeling convenience and accuracy.

RGS is the basis of AeroHydro's MultiSurf (R) surface modeler for IBM-PC compatibles (DOS and Windows) [8]. UNIX workstation versions are under development. All the mathematical and logical functions of MultiSurf are available in an ANSI C library RGKERNEL, for incorporation into third party applications. Patent applications are pending which claim many of the novel features of RGS.

RGS lends itself to a highly interactive, highly visual 3-D design environment for creation, viewing, revision and analysis of models. Largely motivated by marine design needs, RGS provides new solutions to many panelization and gridding problems for CFD and FEM applications.

## RGS FUNDAMENTALS

Definitions:

***entity*** -- an abstract *type* of geometric object, e.g. point, line, plane, B-spline surface
***object*** -- a particular concrete *instance* of an entity; e.g. an absolute point at (0., 1., 2.)

In conventional computer-aided geometric design (CAGD), the nouns *entity* and *object* are applied more or less interchangeably to either of these concepts. In RGS, the two concepts are semantically distinguished by the above defined usage. An *entity* is like a blank form with spaces to be filled in with data such as object name, color, visibility, coordinates, names of other objects. An *object* is like a filled-in copy of that form. RGS entities and objects correspond to the "classes" and "objects" of object-oriented programming terminology.

In RGS, every object -- point, curve, snake, surface, etc. -- has a unique object name. The object name may be used to reference the object whenever it is used in construction of other objects. This is the essential means by which many qualitative relationships are captured. The object name can also be used to select, edit and interrogate the object while constructing and editing a model interactively.

Entities can be broadly classified first according to their dimensionality, and secondarily according to their embedding in other objects. Table 1 is a hierarchical listing of currently implemented RGS entities.

Under Points (0-dimensional entities) there are 4 classes:

3-D points (constructed in various ways)
Magnet - a point embedded in a surface
Beads - a point embedded in a 3-D curve
Rings - a point embedded in a snake

Under Curves (1-dimensional entities) there are 3 classes:

3-D parametric curve
Subcurve -- a curve embedded in a curve
Snake -- a curve embedded in a surface

Under Surfaces (2-dimensional entities) there are 2 classes:

3-D parametric surfaces
Subsurface -- a surface embedded in another surface

Table 1 lists many specific entities in each class, with an explanation of the essential data required for each.

Generally, curves are formed from points, in a variety of ways as indicated in Table 1. For example, the data for a Line is simply the names of two points. The data for a B-spline curve is a list of points specified by their object names. Any type of point object (point, bead, magnet, ring, etc.) will serve as a point, and different point types can be freely mixed.

Surfaces are formed in some cases (e.g. B-spline surface, NURBS surface) from a net of points; usually (e.g. Coons patch, lofted surfaces) from a set of curves; or sometimes (e.g. SubSurface, Fillet) from a set of snakes on one or more underlying surfaces. In each case the supporting point, curve, and snake objects are all specified by using their object names; any kind of point, curve or snake can be used, and these can be freely mixed.

Most of the basic curve and surface entities supported under RGS are well known. The difference lies in how the data is organized and obtained to evaluate the curve and surface objects. Some novel entities have arisen by recognizing gaps in the logical structure of RGS; for example, the B-lofted and BlendCenter surfaces, which are useful generalizations of the B-spline surface and Coons patch.

Six RGS surface entities (Ruled, Arc-Loft, Foil-Loft, B-Loft, C-Loft, X-Loft Surfaces) are classed as "lofted surfaces", and share a common structure. Each lofted surface is supported by 2 or more master curves, depending on the surface type. Evaluation of a single point at parameters u,v on a lofted surface is a 3-stage process:

(1) Evaluate each of the master curves at a parameter value t = u
(2) Construct the curve of appropriate type (Line, Arc, Foil, B-spline or C-spline) which uses these points as data
(3) Evaluate the curve at parameter value t = v

| Surface Entity | No. of MC's | Lofting Curve |
|---|---|---|
| Ruled | 2 | line |
| Arc-lofted | 3 | circular arc |
| Foil-lofted | 3 or 5 | NACA foil |
| B-lofted | >1 | B-spline |
| C-lofted | >1 | interpolating spline |
| X-lofted | >1 | explicit spline |

Figure 1 is an example of an RGS model for a 30-ft sailboat hull. 'hull' is a C-Lofted surface, supported by 3 master curves 'MC1', 'MC2', 'MC3'. Each of the master curves is a B-spline curve, supported by 4 absolute points. The surface is displayed by showing some of the longitudinal "lofting" C-splines.

Another way to view RGS is as a *programming or representation language*. The data base ("Model File") for the model of Figure 1 is given in Table 2 as an example. Each line in the Model File creates an object. The entity names 'AbsPoint', 'BCurve', etc. may be viewed as commands or functions, which are followed by required parameters in prescribed order. The language has a prescribed syntax. Models can be created and modified by editing text files adhering to this syntax. Dependencies are represented by the occurrence of object names as parameters for other objects.

Since the model file contains only concise instructions for building the absolute model, it is *extremely compact* compared with conventional CAD representations. Model files average about 35 bytes per object, whether the object is a point, curve or surface.

Snakes are generally formed from magnets, in a variety of ways as indicated in Table 1. (Rings also qualify as "magnets" for this purpose, since a ring is logically constrained to the surface to which its snake belongs.) The snakes are highly parallel to the curve entities, so they work in a consistent fashion. Snakes can also serve as curves in the construction of surfaces.

"Relative" entities are specified by coordinate or parameter offsets from another object of the same class. For example, a relative bead is specified by a parameter offset dt from a basis bead. It lies on the same curve as its basis bead, at a fixed offset in parameter space. If the basis bead is moved along the supporting curve, the relative bead automatically moves also, to maintain its (parametric) distance away. Use of these relative entities is an important mechanism for capturing and enforcing quantitative relationships between objects.

All curves and snakes are parameterized from 0 to 1 in a parameter t. All parametric surface patches are parameterized from 0 to 1 in each of two parameters u,v, in most cases inheriting the parameterization of their supporting curves. Beads, rings and magnets are located using these parameters. Thus, objects located on a curve or surface will automatically shift to "similar" positions following a change in their supporting curve or surface. Uniformity of the ranges of parameterization is an important key to the interchangeability of various curve, snake and surface types in construction of other objects.

Dependencies of objects on other objects are always represented by references using object names. These take several forms; for example:

(a) a Relative Point depends on its basis point.

(b) a Relative bead depends on its basis bead, and indirectly on the curve that supports both beads.

(c) a Coons patch or lofted surface depends on each of its supporting curves, and indirectly on the points that support them

(d) a B-spline Snake depends on each of its supporting magnets, and indirectly on the surface that supports them, and all the curves and points supporting that surface, etc.

Dependency can extend to an arbitrary number of levels. The dependency structure of a model is representable by a directed graph (digraph), which is sometimes a useful way to contemplate the topological structure of a model. (Figure 2 is the dependency digraph for the model given in Figure 1/ Table 2). The digraph structure can also be used advantageously by the program to selectively update only the objects affected by a given change, i.e., the set of objects that depend directly or indirectly on the particular object that was changed.

It is the capture and maintenance of these dependencies that enables the automatic update feature of RGS, providing a dramatically new paradigm for geometric design. We envisage conventional design, computer-aided or otherwise, as a linear process, a sequence of forward stages, each depending on completion of the prior stage. In sequence, each basic component is designed and frozen, so that other components that depend on it can be designed on a firm foundation. At some point, the whole design is evaluated, and usually found deficient in some respects. At this point, it is very costly to go back to an early stage, altering early components, *because all the intermediate design stages have to be repeated.*. With RGS, the intermediate design steps are captured in the model, and the entire model can be cheaply updated at any time. This is a whole new dimension of freedom in design. Instead of segmenting design into a series of stages, and freezing components in non-optimal states, RGS allows the entire geometric design process to be fluid and integrated end-to-end.

Qualitative model properties which result from the capture and enforcement of these relationships, and are thus conserved under changes in underlying objects, are referred to as "durable" properties.

Avoidance of surface-surface intersection problems is accomplished in several ways:

(1) Two lofted surfaces of the same type and the same number of master curves, whose master curves share endpoints, will join accurately and durably along their common edge.

(2) RGS supports several surface entities that interpolate 2 or 4 arbitrary boundary curves. Two such surfaces A and B that share the same curve as data will have a durable common edge along that curve.

(3) Suppose surface B is required to join surface A accurately, not along an edge of A, but somewhere out in the middle of A. This can be accomplished durably by using a snake on A as an edge curve for B, then using a surface for B that interpolates this edge curve.

Although the above methods eliminate the need for intersection and trimming in the great majority of situations, there still arise some modeling situations in which projection or intersection entities are required. These are met by a set of "projection" and "intersection" entities; for example, a ProjSnake is the projection of a curve onto a surface; an IntBead is at the intersection of a curve with a plane or surface. The projection and intersection entities involve iterative solutions, and are therefore subject to all the ills which intersections are heir to: possible nonexistence; possible multiple solutions; possible failures to converge; computational expense, etc. However, under RGS these evils are substantially alleviated by the possibility of providing good starting values for the iteration. For example, the data for an IntBead can be a bead (which identifies the intersecting curve, and provides a starting value for t) and a magnet (which identifies the intersecting surface, and provides starting values for u and v).

## PARAMETRIC DESIGN

"Parametric design" is a capability to specify a family of related designs having N numeric parameters xi, i=1...N, so that the specification of suitable values for all N parameters produces a valid instance of the family. The parameters xi span an n-dimensional "design space" within which, for example, an optimal design can be sought [9].

In RGS, a high degree of separation between topological and metrical components of geometry provides a strong capability for generating parametrically varied models having durable topology and other durable shape characteristics. Essentially, each real number occurring in the model file is a potential parameter; for the most part, these are the X,Y,Z coordinates or offsets of points, or the parameter values of beads, magnets and rings. The non-numeric content of the model file, especially the specification of interobject relationships through use of object names, conveys the topology.

A stronger parametric capability currently under development arises from permitting use of numeric variables and expressions in the model file. This is currently being simulated by use of model file preprocessors.

## PANELIZATION USING PATCHES AND SUBSURFS

Since all parametric surface objects in RGS are topologically rectangular patches, it is quite simple and direct to use the mesh of isoparametric lines on any fully wetted patch to generate quadrilateral or triangular panels. The subdivision of any surface into a mesh is part of its specification, and can easily be changed to control panel counts.

Panel size variations, e.g. cosine spacing on foil elements, is handled through a general facility for reparameterization ("relabeling") of curves and lofted surfaces. The surface mesh division responds to the relabeling of its supporting curves. Figure 3 shows some panel distributions available through relabeling of a wing surface.

Often, there are portions of a base surface object which are not wetted (e.g., the portion of a fuselage covered by the wing root or wing root fillet), and which should therefore not be panelized. The embedded SubSurf provides a general solution for this situation: the wetted portion is covered by quadrilateral SubSurf patches. It is generally possible, in fact easy, to durably coordinate the subdivisions of adjacent patches, e.g. wing and fuselage, so that neighboring panels on the two surfaces have identical corner points, since the two patches share snakes as common boundaries. Figure 4 shows a filleted wing-body example with this property.

## EXAMPLES OF PARAMETRIC DESIGN AND PANELIZATION

### Tension-leg Platform Example

Figure 5 shows several members of a 4-parameter family of tension-leg platforms (TLP's). TLP's are buoyant structures, moored to the seafloor by taut cables, commonly used for offshore drilling platforms in relatively deep water. The vertical cylinders are called "legs" and the horizontal members are "pontoons". Panel methods are used to analyze the response of TLP's and their mooring systems to gravity waves in the ocean environment.

In this model family, both legs and pontoons are circular cylinders; the 4 parameters are:

leg depth
leg radius
pontoon depth
pontoon radius

Each parameter is associated with a single real number in the model file. Figure 5 shows the base model and 4 variants, with one parameter varied at a time.

The essential structure of the model is as follows:

(1) Exploiting symmetry, only 1/8 of the model needs to be explicitly constructed.
(2) The model begins with a base half-cylinder (RevSurf) for the complete leg surface.
(3) A circle is constructed to define the pontoon depth and radius.
(4) The circle is projected horizontally onto the leg, making a ProjSnake.
(5) The pontoon surface is a ruled surface between the circle and the ProjSnake.
(6) The wetted portion of the leg (outside the ProjSnake) is cut by snakes into 3 SubSurfs.
(7) The bottom is a RevSurf made from a radial line as meridian.

Note the durable neat join between neighboring panels across the pontoon-leg and bottom-leg junctions, detailed in Fig. 6.

## Sailing Yacht Example

Figure 7 shows a panelization of the underwater portions of a sailing yacht hull, keel and rudder. The yacht hull provides buoyant support for the vessel; the keel provides horizontal lift to resist side forces from the sails; the rudder provides directional control and stability. The lift and induced drag performance of the combination are crucial to sailing performance. These characteristics are typically analyzed by panel-based potential flow solutions.

In this example, the keel, keel tip, rudder, and rudder tip surfaces are fully wetted patches. A problem typical of panel methods in hydrodynamics is that only a portion of the hull surface is wetted, i.e., the portion *below* the water level and *outside* the keel and rudder root areas. This is handled with a single SubSurf covering this area. The edge of the SubSurf along the waterline is an intersection snake; the edge along the centerplane is a PolySnake which follows the centerplane where possible, but is indented around the keel and rudder. The SubSurf has one degenerate edge (a magnet) at the aft ending of the waterline, producing one row of triangular panels at this corner. The neat join between neighbor panels at the keel-hull and rudder-hull junctions arises from use of snakes on the hull surface to define both the root junctions and the centerline PolySnake. These properties are durable with respect to many possible variations in the hull, keel, and rudder geometries.

### Submarine Example

Fig. 8 shows the base model and several variants on a parametric submarine design. The hull is a body of revolution; the conning tower or sail is a B-lofted surface attached to a snake on the hull. Available parameters include:

hull diameter
bow and stern prismatic coefficients
nose radius
sail height
sail length
sail width
sail longitudinal position.

The lower hull is a fully wetted patch; the wetted portion of the upper hull is a SubSurf built around the snake at the sail-hull junction. The panelization is topologically invariant with respect to all shape parameters, except the longitudinal position of the sail; if this changes too much, there needs to be some reparametrization of the dorsal PolySnake to avoid skewed panels.

### Propeller Example

Fig. 9 shows a member of a parametrically generated propeller series. The data for this example is a standard propeller sheet (Table 3) giving chord, thickness, camber, rake and skew at selected radii. An auxiliary program performed curve fitting and smoothing on the radial variations of these shape parameters, generating a MultiSurf model file which reproduces the specified blade geometry. The modeled blade is a B-lofted surface using FoilSnakes on concentric cylinders as master curves.

## VOLUME GRIDDING USING RG SOLIDS

A straightforward extension of RGS to 3-D parametric solids has been made at a pilot level. This appears to have important potential for 3-D gridding. A parametric solid is a vector function of three parameters: $\mathbf{x} = \mathbf{x}(u,v,w)$. It is easy to formulate RG solids constructed in several ways analogous to various surface entities; for example:

RuledSolid -- linear interpolation between two surfaces
BSolid -- trivariate tensor product using a 3-D array of control points
CLoftSolid -- spline interpolation through 2 or more surfaces
BlendSolid -- trivariate linear blending of 6 boundary surfaces
TranSolid, SweepSolid -- a surface patch swept along an axis curve

A parametric solid can easily be subdivided along isoparametric surfaces to create a structured mesh. It appears that the same techniques that produce neat, durable joins between neighboring patches in 2-D will extend naturally to solids, allowing continuity of elements between adjacent blocks.

## CONFIGURATION OPTIMIZATION

As diagrammed in Fig. 10, MultiSurf can serve as a geometry and panelization engine in a closed-loop optimization system including an optimizer and one or more analysis programs. The optimizer treats the model as a point in design space, and feeds design-space parameters to a model file generator. The resulting model file is read into MultiSurf , which generates a standard output file

of surface meshes. A preprocessor casts this mesh data into the particular format required for input to the analysis program. The analysis program is run, and its output file is inspected by a postprocessor which extracts the required performance measures and returns the value of the objective function to the optimizer.

## CONCLUDING REMARKS

We have outlined the principles of Relational Geometric Synthesis and given several examples of application to panelization and gridding problems. The capability of RGS to define parametric families of topologically invariate designs and to generate geometric representations of them, including high quality panelizations for analysis, promises large reductions in the skilled labor currently required for application of CFD and FEM in practical engineering design.

## ACKNOWLEDGMENT

This work was supported in part by the U. S. Navy, Carderock Naval Surface Warfare Center, under contract N00167-95-C-0003.

## REFERENCES

1. Letcher, J. S. Jr. and Shook, D. M.: "Applications of Relational Geometric Synthesis in Sailing Yacht Design", *Proceedings 11th Chesapeake Sailing Yacht Symposium*, S.N.A.M.E. (1993)

2. Letcher, J. S. Jr., Shook, D. M. and Shepherd, S. G.: "Relational Geometric Synthesis -- Part 1: Framework", *Computer Aided Design* (accepted for publication)

3. Armit. A.: "TAG -- A High Performance Interactive 3D Graphics System", *Computers in Industry*, vol. 3, pp. 117-123 (1982)

4. Snepp, D. K. and Pomeroy, R. C.: "A Geometry System for Aerodynamic Design", AIAA-87-2902 (1987)

5. Capron, W. K. and Smit, K. L.: "Advanced Aerodynamic Applications of an Interactive Geometry and Visualization System", AIAA-91-0800 (1991)

6. Gentry, A. E.: "Requirements for a Geometry Programming Language for CFD Applications", NASA CP-3143 (1992)

7. MacKrell, J.: "Making Sense of a Revolution", *Computer Graphics World*, v. 16, no. 11 (Nov. 1993)

8. AeroHydro, Inc.: *MultiSurf User's Manual* (1994)

9. Haug, E. J. and Arora, J. S.: *Applied Optimal Design*, Wiley, New York (1979)

Table 1. MultiSurf entities

Points (0-dimensional), including:

- 3-DOF points: Absolute Point -- X,Y,Z coordinates
  - Relative Points -- X,Y,Z offsets from another point
  - Polar coordinates relative to another point
- 2-DOF points: Magnets -- points constrained to lie on a surface
  - Absolute Magnet -- u,v parameters
  - Relative Magnet -- u,v offsets
  - Projected Magnet -- projection of point onto surface
  - Intersection Magnet -- intersection of surface with a curve
- 1-DOF points: Beads -- points constrained to lie on a curve
  - Absolute Bead -- t parameter
  - Relative Bead -- t offset
  - Intersection Bead -- intersection of curve with plane or surface
  - Rings -- points constrained to lie on a snake
    - Absolute Ring -- t parameter
    - Relative Ring -- t offset
    - Intersection Ring -- intersection of snake with plane or curve
- 0-DOF points: Projected Point -- point projected onto a plane
  - Mirror Point -- point reflected in a plane
  - Offset Point -- offset along normal to surface
  - Tangent Point -- offset along the tangent to a curve

Curves (1-dimensional), including:

- Line -- 2 points
- Circular Arc -- 3 points
- Conic sections -- type, 3 points, 2 parameters
- NACA Foils -- 3 points (half section) or 5 points (full section)
- B-spline Curve -- type, N points
- C-spline Curve -- type, N points
- X-spline Curve -- type, N points, end slopes or moments
- NURBS Curve -- type, knotlist, N x ( point, weight )
- SubCurve -- portion of a curve between 2 beads
- PolyCurve -- concatenation of N curves
- Relative Curve -- curve, 2 points, graph
- Projected Curve -- curve projected onto a plane
- Mirror Curve -- curve, plane
- Offset Curve -- snake, 2 offsets, graph

Snakes (1-dimensional) -- curves constrained to lie on a surface:

- Line Snake -- 2 magnets
- Arc Snake -- 3 magnets
- Foil Snake -- 3 magnets (half-section), or 5 magnets (full section)
- B-spline Snake -- type, N magnets
- C-spline Snake -- type, N magnets
- NURBS Snake -- type, knotlist, N x ( magnet, weight )

Snakes continued:

SubSnake -- portion of a snake between 2 rings
PolySnake -- concatenation of N snakes
Relative Snake -- snake, 2 magnets
EdgeSnake -- type, surface
Projected Snake -- projection of a curve onto a surface
Intersection Snake -- intersection with a plane or surface

Surfaces (2-dimensional), including:

Revolution Surface -- curve, axis line, 2 angles
Translation Surface -- 2 curves
Coons patch -- 4 curves (nominally end-to-end), 2 graphs
Developable Surface -- 2 curves
Ruled Surface -- 2 curves
B-spline Lofted Surface -- type, N curves
C-spline Lofted Surface -- type, N curves
X-spline Lofted Surface -- type, N curves, 4 graphs for end conditions
Foil-lofted Surface -- type, 3 or 5 curves
B-spline Surface -- utype, vtype, N x M points
NURBS Surface -- utype, vtype, 2 knotlists, N x M x (point, weight)
Offset Surface -- surface, 4 offsets
Relative Surface -- surface, 4 points
Blister -- type, 2 snakes, curve
Fillet -- 2 snakes on each of 2 surfaces
SubSurface -- 4 snakes (nominally end-to-end)
Projected Surface -- surface, plane
Mirror Surface -- surface, plane
Sweep Surface -- surface, curve
PolySurf -- concatenation of N surfaces

Miscellaneous:

Planes -- nonparametric planes, specified several ways (e.g., 3 points)
Contours -- set of parallel plane, cylindrical, or spherical sections
Knotlist -- knot vector, used in NURBS entities
Wireframe -- name of a wireframe file, point
Graphs -- univariate function of parameter
Frames -- local orientable coordinate system

Table 2. Model file for the example of Figure 1.

```
MultiSurf 0.35 /y
Rem demo of C-lofted surface with 3 B-spline MCs
AbsPoint  P11  14 1   0.000    0.000    3.600;
AbsPoint  P12  14 1   1.367    0.000    0.602;
AbsPoint  P13  14 1   2.324    0.000   -0.800;
AbsPoint  P14  14 1   3.000    0.000   -0.900;
BCurve  MC1  11 1  10 *  2
  { P11  P12  P13  P14 };
AbsPoint  P21  14 1  15.000    4.815    2.560;
AbsPoint  P22  14 1  15.000    5.046    0.628;
AbsPoint  P23  14 1  15.000    3.603   -0.870;
AbsPoint  P24  14 1  15.000    0.000   -1.175;
BCurve  MC2  11 1  10 *  2
  { P21  P22  P23  P24 };
AbsPoint  P31  14 1  30.000    3.500    2.760;
AbsPoint  P32  14 1  30.000    3.500    1.320;
AbsPoint  P33  14 1  30.000    2.500    0.120;
AbsPoint  P34  14 1  30.000    0.000    0.120;
BCurve  MC3  11 1  10 *  2
  { P31  P32  P33  P34 };
CLoftSurf  hull  10 1  10 1 20 1  0 3
  { MC1  MC2  MC3 };
XContours  stations  12 1  0 10  1.8 2.735;
ZContours  waterlines  13 1  0 0  0. 1.;
EndModel
```

Table 3. Data sheet for parametric propeller design.

DESIGN DATA PROPELLER 0000 -R.H.
DIAMETER = 12.00 INCHES SHIP ( 304.8 MILLIMETRES SHIP )

| R/R | P/D | CHORD/D | T/D | CAM/D | AFT. RAKE/D | AFT. RAKE (DEGREES) | PROJECTED SKEW (DEGREES) | R/R |
|---|---|---|---|---|---|---|---|---|
| .200 | 1.5358 | .1735 | .0434 | .00583 | 0.00000 | 0.00 | 0.00 | .200 |
| .300 | 1.5125 | .2283 | .0357 | .00933 | 0.00000 | 0.00 | 9.29 | .300 |
| .400 | 1.4592 | .2750 | .0294 | .01058 | 0.00000 | 0.00 | 18.79 | .400 |
| .500 | 1.3858 | .3125 | .0240 | .01067 | 0.00000 | 0.00 | 28.01 | .500 |
| .600 | 1.2958 | .3375 | .0191 | .00950 | 0.00000 | 0.00 | 36.75 | .600 |
| .700 | 1.1975 | .3475 | .0146 | .00800 | 0.00000 | 0.00 | 45.43 | .700 |
| .800 | 1.0958 | .3342 | .0105 | .00633 | 0.00000 | 0.00 | 54.23 | .800 |
| .900 | .9958 | .2808 | .0067 | .00442 | 0.00000 | 0.00 | 63.11 | .900 |
| .950 | .9483 | .2192 | .0058 | .00317 | 0.00000 | 0.00 | 67.76 | .950 |
| .960 | .9383 | .2021 | .0054 | .00292 | 0.00000 | 0.00 | 68.72 | .960 |
| .970 | .9292 | .1800 | .0053 | .00250 | 0.00000 | 0.00 | 69.68 | .970 |
| .980 | .9196 | .1521 | .0051 | .00208 | 0.00000 | 0.00 | 70.65 | .980 |
| .990 | .9108 | .1146 | .0050 | .00183 | 0.00000 | 0.00 | 71.61 | .990 |
| 1.000 | .9013 | 0.0000 | 0.0000 | 0.00000 | 0.00000 | 0.00 | 72.59 | 1.000 |

| | |
|---|---|
| SECTION CAMBER DISTRIBUTION | NACA A=0.8 MEANLINE |
| SECTION THICKNESS DISTRIBUTION | NACA 66 (TMB MODIFIED) |
| EXPANDED AREA RATIO | .7253 |
| PROJECTED AREA RATIO | .5701 |
| MEAN WIDTH RATIO | .2848 |
| BLADE THICKNESS FRACTION | .0488 |
| RAKE ANGLE AT BLADE TIP | 0.0000 (DEGREES) |
| PROJECTED SKEW ANGLE AT BLADE TIP | 72.5880 (DEGREES) |
| LINEAR RATIO | 1.0000 |

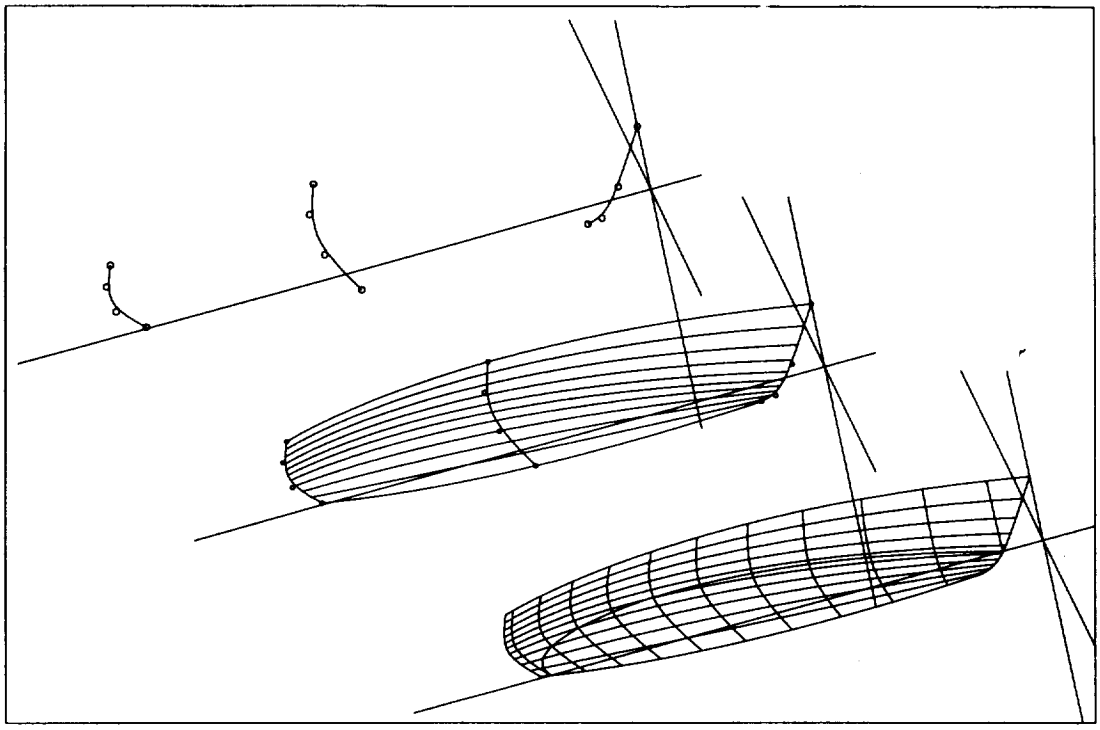


Fig. 1. A C-lofted hull surface with 3 B-spline master curves, each supported by 4 points.

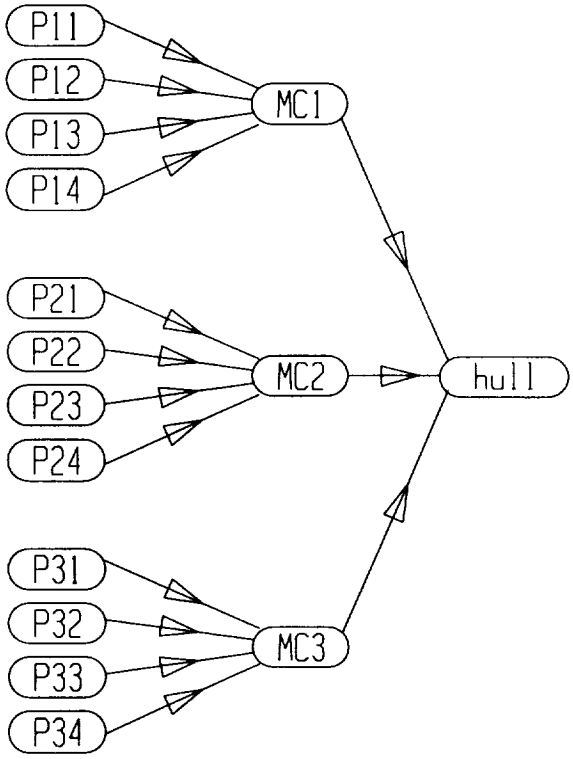



Fig. 2. Digraph of object dependencies for the hull example in Fig. 1/Table 1.

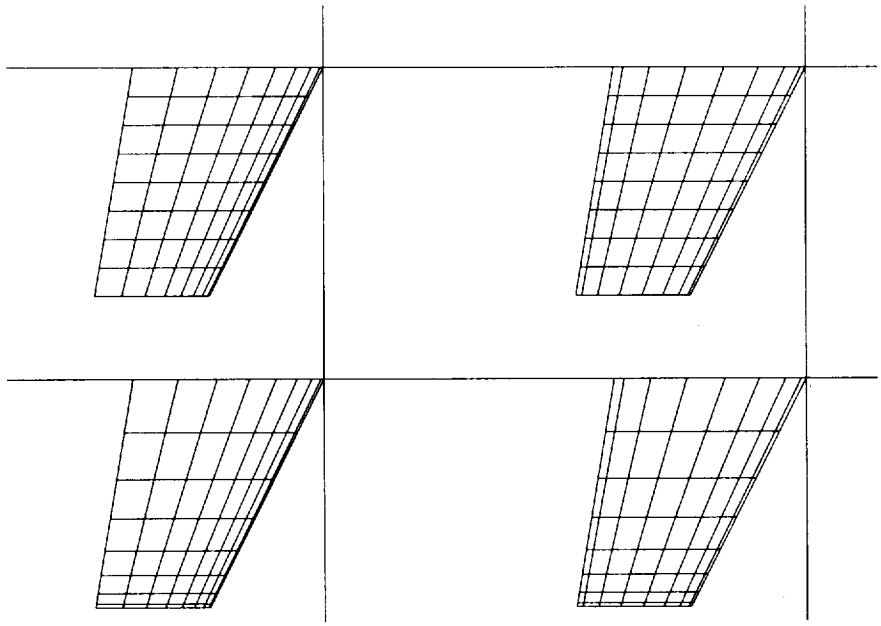


Fig. 3. Typical variations in spanwise and chordwise panel distributions available by relabeling the supporting curves of a wing surface.

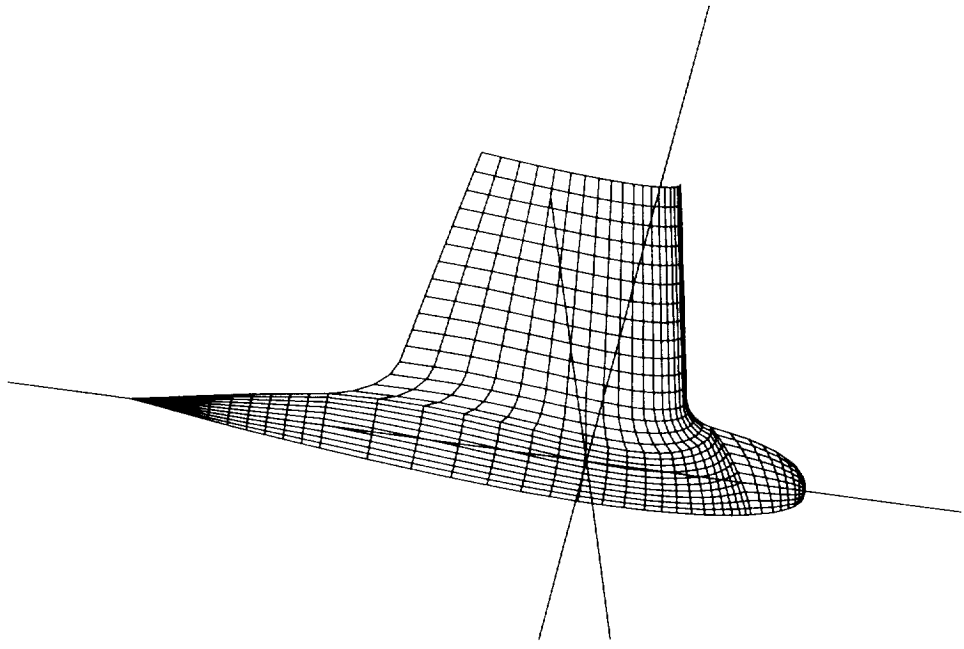


Fig. 4. Panelization of a filleted wing-body junction using SubSurfs on the wing and body.

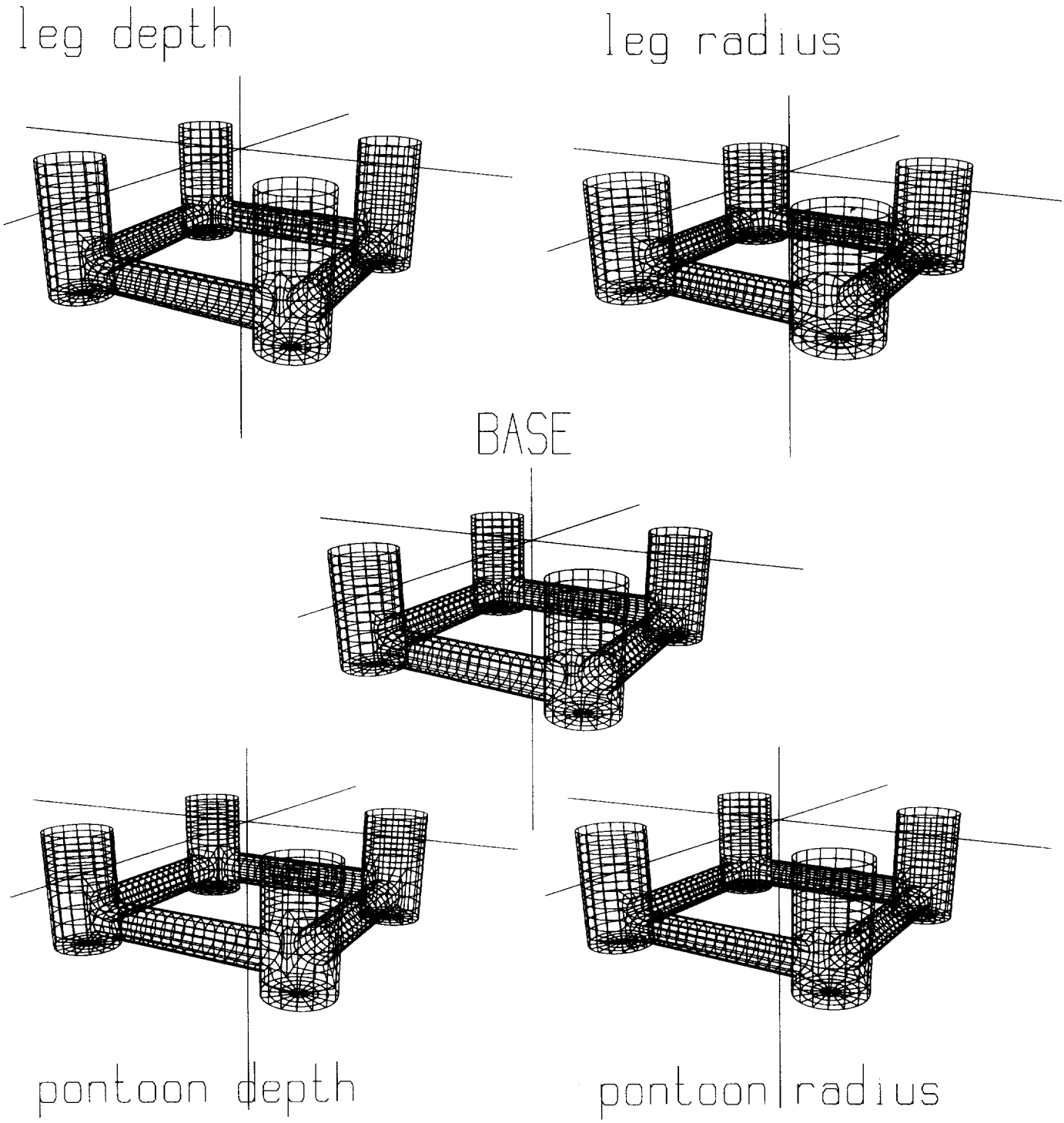



Fig. 5. Parametric variations in tension-leg platform geometry and panelization. Topologically similar panelizations are produced automatically as geometry is parametrically varied over wide ranges.

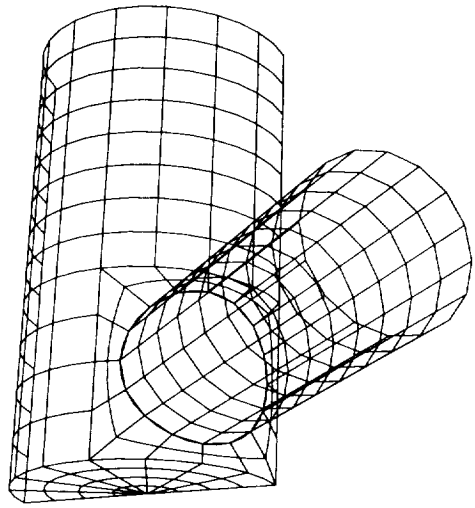


Fig. 6. Detail of a tension-leg platform panelization showing neat joins between pontoon, leg, and bottom panels.

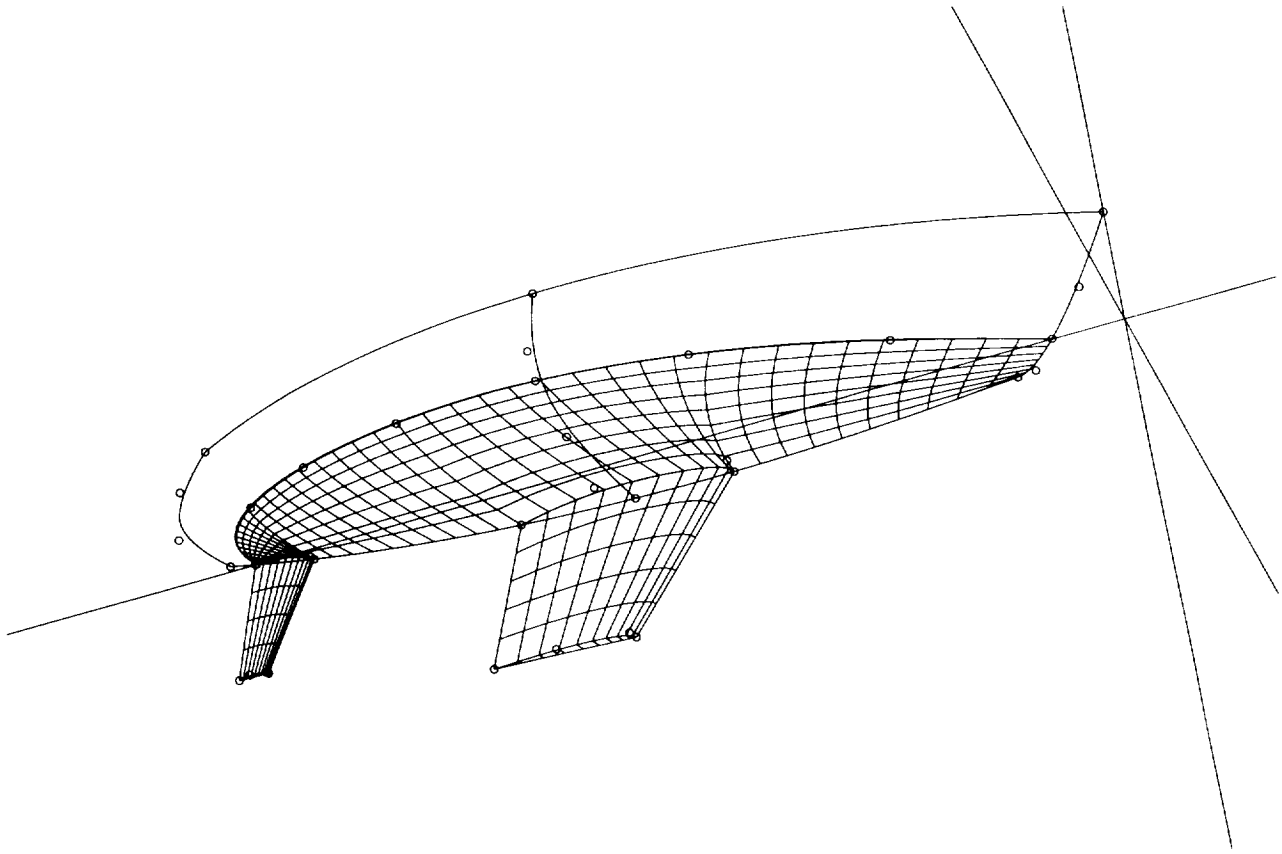


Fig. 7. Panelization of sailing yacht hull, keel, and rudder combination. The hull panels are a SubSurf between an intersection snake along the waterline and a PolySnake along the ventral edge.

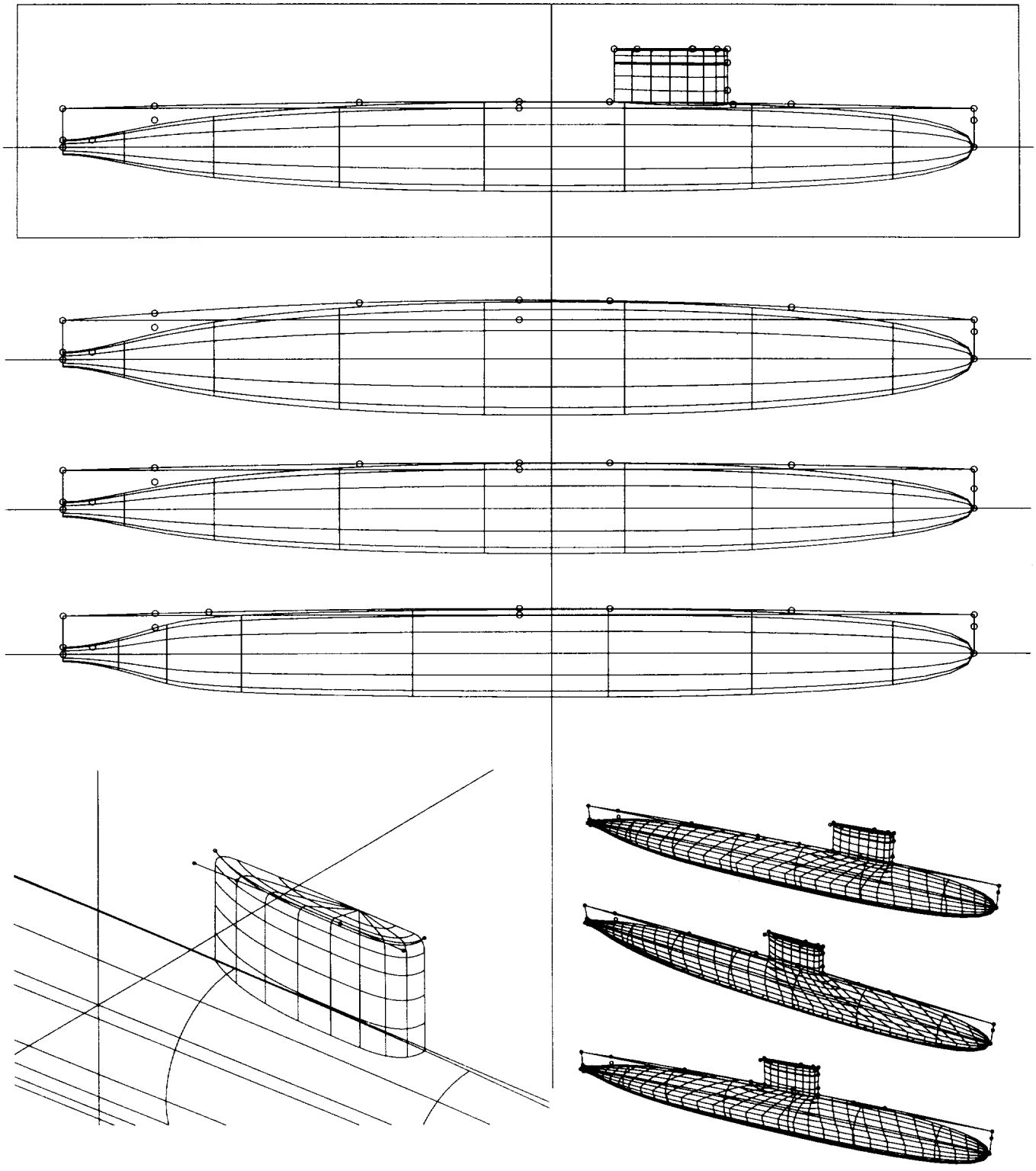


Fig. 8. Parametric submarine example, showing hull and sail variations, with instant repanelization by SubSurfs.

N95-28772

# EXAMPLES OF GRID GENERATION WITH IMPLICITLY SPECIFIED SURFACES USING GridPro$^{TM}$/az3000 [1]: FILLETED MULTI-TUBE CONFIGURATIONS

Zheming Cheng and Peter R. Eiseman

Program Development Corporation
300 Hamilton Avenue, Suite 409
White Plains, NY 10601

## SUMMARY

With examples, we illustrate how implicitly specified surfaces can be used for grid generation with **GridPro/az3000**. The particular examples address two questions: (1) How do you model intersecting tubes with fillets? and (2) How do you generate grids inside the intersected tubes? The implication is much more general. With the results in a forthcoming paper which develops an easy-to-follow procedure for implicit surface modeling, we provide a powerful means for rapid prototyping in grid generation.

## INTRODUCTION

The theory for implicit surface formulations has its roots in algebraic geometry and asymptotics. The implicit form, as opposed to explicitly defined parametric surfaces, is given by a function $U$ of the embedding space variables: the Cartesian $x$, $y$, and $z$. The surface definition is then a level surface of the function $U(x, y, z)$. A level surface is just the set of points for which the specified function is a constant. Level surfaces, for example are commonly used to graphically display the contour lines of some variable. Since we are interested in only one level surface, there is no loss of generality in forming $U(x, y, z)$ in such a way that our desired surface appears when $U(x, y, z)$ is 0. This can always be made to happen for if the desired surface appeared with a value of $U$ equal to another constant $C$, then we would merely replace $U$ by $U - C$. Geometric modeling with implicit surfaces has been considered by [2].

As with contour lines in graphics, it is clear that the surface can have a richer topology. In the case of intersecting tubes, which serves as an example to illustrate the use of implicit surfaces, the topology is such that the surface has holes at the openings of each tube. Unlike the parametric constructions, this surface cannot be smoothly contracted to a point: it has non-trivial homotopy and cohomology groups. The parametric constructions such as NURBS surfaces are not general enough. They can only be applied in a local piecewise sense: not a global sense.

To explore fully the potential of implicitly specified surfaces in grid generation, there are two important aspects need to be addressed: 1). How can a given implicit surface be used by a grid generator. This is the topic of this paper. 2). How can a user setup implicit surfaces for his/her particular application with ease. This aspect falls in the general area of what can be called implicit surface modeling. This will be addressed in a forthcoming paper, where a well defined and easy-to-follow procedure for general implicit surface modeling is developed.

Altogether, implicit surfaces represent a very useful and powerful facility to specify geometries.

In many situations, it is more advantageous than the explicitly (parametrically) defined surfaces. Often, a simple function can represent a rather complex geometry as is the case shown in this paper. It can be used to define generic shapes of surfaces. Therefore, a grid generator that accepts implicit surfaces is very well suited to perform rapid prototyping in grid generation.

The general implicit surface definitions by level surfaces is one of the surface types that can be directly taken by the grid generator, **GridPro/az3000** [3, 4], for either global or local surface specifications. In fact, for any grid generator to be considered really advanced must be able to deal with implicit surfaces efficiently.

**GridPro/az3000** is the multiblock grid generator with automatic zoning. Aside from surface geometry, the primary user input is the pattern of grid lines or surfaces. This pattern is referred to as the grid topology. The definition of grid topology is given by a topology input language (TIL). This is also used to specify the number of grid points and clustering criteria. Once the TIL code is in place, the grid is generated as a batch job in a manner which is steered by a dynamic schedule file. The latter file allows the user a great deal of flexibility. For example, rather than just run the case and write out the results, the user may desire to start with fewer grid points or lesser clustering and then to gradually increase at latter steps in the evolution toward convergence. A common motivation is for more efficiency when only modest computational resources are available for the size of problem at hand.

Our discussion will start with the implicit geometry modeling for filleted intersecting tubes and will continue with the development of TIL code for the generation of high quality (smooth and nearly orthogonal) grids inside the multi-tube configurations. Examples will be given to show the variety of configurations, the grid and surface quality, and the ease with which these items can be created and presented for analysis. The multiplely branched tubes are frequently encountered surfaces in geometry modeling and grid generation for a variety of different fields. As a natural consequence, the mathematical formula for the surface model should be of considerable interest on its own right.

## MULT-TUBE GEOMETRY MODELING

The question of modeling the geometry of tube intersections has been addressed before in a piecemeal fashion. The typical approach is to first find the intersection between the tubes. Once the intersection is known, the next step is to move away from that curve of intersection to create displaced curves on each tube at some reasonable distance from the intersection. This displacement is most often formed moving a fixed distance along geodesic paths that emanate orthogonally from the curve of intersection. With these displaced curves along each tube, the fillet is formed by a curve of interpolation between them. This starts on one tube with the displaced position and available geodesic tangent direction and ends on the other with the same conditions. When taken together, a collar type surface is formed. The collar connects the tubes with first derivative continuity. This process requires the solution of a surface surface intersection problem, the computation of geodesic distances to produce the displaced curves, and the act of interpolation to construct the collar surface. For this work, there are three surfaces required to define the intersection of two tubes in a T-type configuration. The result is only first derivative continuous. For each application of this technique, a collar is added. For example, with full tubes along the each coordinate axis, there would be 12 surfaces (the tubes for $x$, $y$, $z$, $-x$, $-y$, and $-z$ plus 6 collars). Moreover, should a tube aspect ratio or squareness be added, then the associated problems of construction would be more intensive.

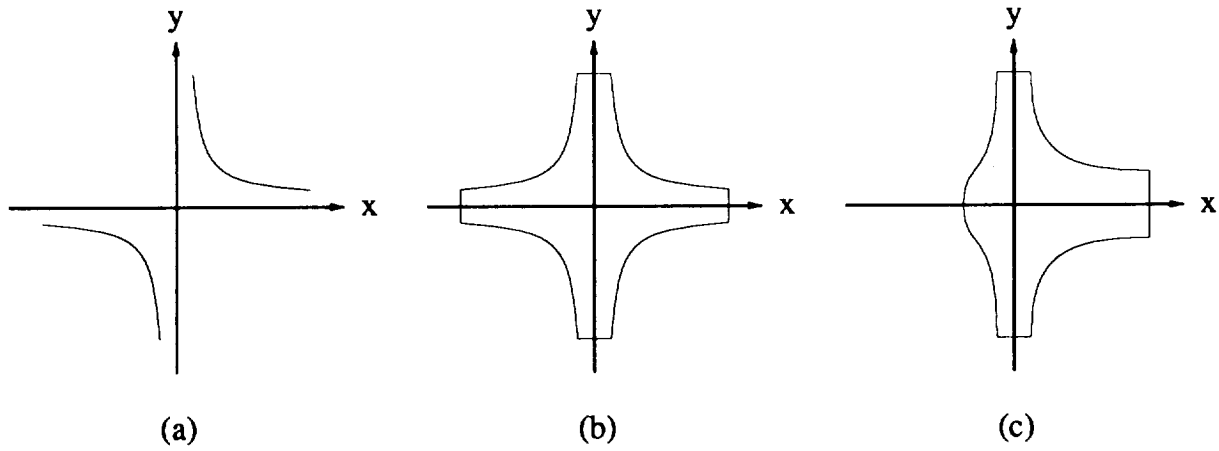



Figure 1: (a). The zero set of $U = 1 - xy$ is shown. (b). The merge of all axial channels are shown. The curved walls are given by either $U = 1 - x^2y^2$ or the more general $U = a + bx^2 + cy^2 - x^2y^2$. (c). The coefficient $b$ is blended from -1 along $x < 0$ to 1 along $x > 0$.

Unlike the traditional means for fillet creation, we offer a technique which does it all with one infinitely differentiable surface. Moreover, each tube can be assigned a cross-sectional squareness and aspect ratio.

**Two dimensional models**

Our examination of the geometry modeling process will start with two dimensions and will lead by intuition. This should provide simplicity and clarity to the discussion of the implicit methodology. In keeping with this objective, our first problem is to attempt to define the geometry of four channels which follow the $(x, y)$ coordinate axes and come together at the origin. If we examine the function $y = 1/x$ , then immediately we have two curves which asymptote to the axes. As we go towards plus and minus infinity, it approaches the four semi-axes. Unfortunately, there are only two curves associated with this function. One is in the first quadrant while the second is in the third quadrant. Thus, only one half of each channel can be bounded. In order to correct this deficiency, we note that general shape is close to what we want, but that we need to also get the same curves on the opposite side of each semi-axis. This can be readily accomplished by simply taking the square of the equation that we started with. This means that both the original equation $y = 1/x$ as well as the additional equation $y = -1/x$ are satisfied when we use $x^2 = 1/y^2$. It is the additional equation that inserts the previously missing curve components for the second and fourth quadrants. The result is symmetric as is the equation. These are illustrated in Fig 1 (a) and (b).

The above equations between $x$ and $y$ can also be readily expressed in the form of $U(x, y) = 0$. In particular, the symmetric equation is defined with $U(x, y) = 1 - x^2y^2$. Alternatives are provided by a multiplication with any strictly positive or negative quantity. Such multiplication's will not change the 0 set of $U(x, y)$. That is, the level curves at constant 0 as defined by $U(x, y) = 0$ will be the same. With the current choice, the value along each axis will be unity. This will decay as the axial regions are departed. The decay will continue through the level curves that bound the channels and be negative outside of them. Because of this continuous decay, the gradient of $U(x, y)$ will be pointing into the channels from each point on the channel walls. As we will see, a requirement for the surface definition is that it be oriented in such a way that the wall normal

vectors point into the region to be gridded. Thus, this choice of sign, will at least, possess the convenience of providing the gradients with the preferred orientation. To complete the definition of the physical region, each channel must have an end. This is easily accomplished by specifying four straight lines which successively cap off each channel with a perpendicular slice and with normals that are oriented into the physical region. In level surface form, a straight line with normal $(a, b)$ is given by $U(x, y) = ax + by + c$ (Fig 1 (b)). Should a surface be given where the normal is not in the correct direction, then a reversal of orientation can also applied in the TIL code with the flag "`-o`".

As one might wish to make the channels arbitrarily long with constant cross-section, a deficiency immediately appears. That is because the channel width is decreasing at a rate inversely proportional to the distance away from the origin. A partial solution to this problem is to insert a parameter $a$ into the level surface definition to arrive at $U(x, y) = a - x^2y^2$. Then, at least, the value of $a$ can be chosen relative to the distance from the origin. This will assure that the cross-section at the cap is not too close to 0. However, it will also mean that the channels may be too fat at closer distances. Thus, a more substantial correction is needed. For this purpose, the function $U$ is generalized to admit a more desirable asymptotic behavior. It is given by

$$U(x, y) = a + b \cdot x^2 + c \cdot y^2 - x^2 \cdot y^2 \tag{1}$$

The strategy, here, is to add pure quadratic terms in both $x$ and $y$ so that as either $x$ or $y$ becomes large near an axis, one of the new terms will become significant to the same order of the last term in the expression. Since each of the 4 channels behaves in the same manner, we need to examine only one to see what is happening. Thus, we will focus our attention on the positive $x$-axis. Specifically, if the position vector $\mathbf{r} = (x, y)$ is moved away from the origin but stays in the region near the positive $x$-axis, then the only significant term in $U(x, y)$ is $x^2(b - y^2)$. Thus, the channel "radius" is nearly $y = \sqrt{b}$ which means that the channel cross-section is nearly twice that. In a more analytical sense, we can rearrange the equation $U(x, y) = 0$ to cast it into the form

$$y^2 - b = \frac{a + c \cdot y^2}{x^2} \tag{2}$$

Upon examination, the numerator on the right hand side is clearly bounded since the position vector lies near the x axis which in turn implies that y cannot become arbitrarily large. However, the denominator does become arbitrarily large as we go arbitrarily far along the positive $x$-axis. As a consequence, the right hand side becomes progressively smaller as we travel along the positive $x$-axis. With this clear observation, we see that the equation rapidly reduces to something like that of setting the left hand side to 0. The error term (call it $E$) in this process is then the right hand side. It tells us how much deviation there is from our desired asymptotic lines. The asymptotic lines are at $y = \sqrt{b}$ and $y = -\sqrt{b}$. As the channel boundaries emerge from the juncture near the origin, they approach the asymptotic lines from the outside. To insure that they stay on the outside, E must be positive for the actual channel "radius" is $y = \sqrt{b + E}$. This positivity condition then becomes a condition on the parameters which is that $a + bc$ must be positive. With $\sqrt{b}$ and $\sqrt{c}$ given as the radii of the channels along the respective $x$ and $y$ axes, the condition really becomes a condition upon the choice of $a$ (Fig 1 (b)).

At this stage, we have successfully constructed a level surface function for the 2D case of four channels which smoothly come together at the origin. The next step is to see if it can be generalized to a system of fewer channels. For this purpose, we consider the basic form of $U(x, y)$ that was already seen to be successful. Upon examination of the last equation where the right hand side was considered to be a positive error term $E$, the prior solution would not exist if $b$ were negative

rather than positive. With a constant negative value for $b$, the left hand side is strictly positive and bounded away from 0 while the right hand side approaches 0 with larger values of $x$. This means that the equation cannot be satisfied for large x. Thus, it must be bounded in $x$. In fact, by solving for $x$ as a function of $y$ we get

$$x = \pm\sqrt{\frac{a + cy^2}{-b + y^2}} \tag{3}$$

which is certainly well defined when $b$ is negative. This function gives the contour on either side of the $y$-axis. It is symmetric about the $x$-axis and crosses it with a value of $\pm\sqrt{\frac{a}{-b}}$. The asymptotic lines are located at $\pm\sqrt{c}$. Thus, the contour will bulge out a distance of $\sqrt{\frac{a}{-b}} - \sqrt{c}$ as it crosses the $x$-axis. If $a = -bc$, then there is no bulge. This can also be seen by a direct substitution into the above equation since it then reduces to $x = \pm\sqrt{c}$. The result is that the channel walls are exactly the asymptotes.

As we have just witnessed, a negative value of the parameter b means that there is no tube in the corresponding $x$-axis directions. Unfortunately, it kills off the channels on both the positive and negative $x$-axis at once. This leaves the rather uninteresting case of a single straight channel. To get to the next level of interest, we need to have a channel on one side but not the other. We shall thus consider the case for a channel along the positive $x$-axis which fillets into a channel along the entire $y$-axis. This will require $b$ to be negative along the negative $x$-axis and positive along the positive $x$-axis. It also means that $b$ must depend upon $x$ as opposed to its previous role as a constant. Since it still represents the square of the channel radius, it must, at the very least, approach a constant value with increasing $x$. This will then assure us of a channel with fixed cross-section. To achieve that fixed state within the physical region under consideration, it is important to have the approach be sufficiently rapid. With these motivational facts, we need to use an asymptotic blending function with a rate which can be controlled. An ideal candidate for this purpose is the hyperbolic tangent. To allow all possibilities, we shall write $b$ in the form

$$b = p \cdot f(x) + q \cdot [1 - f(x)] \tag{4}$$

where $f(x) = [1 + \tanh(wx)]/2$

The allowed constants along the respective positive and negative $x$-axis are respectively $p$ and $q$. To consider the case of interest, the value of $q$ is negative while the value of $p$ is positive. This means that we will only have a channel of radius $\sqrt{p}$ along the positive $x$-axis. As the blending function $f(x)$ leaves the origin, it approaches 0 along the negative $x$-axis and 1 along the positive $x$-axis. The rapidity with which it approaches each value is controlled by the damping factor $w$. (see Fig 1 (c)). While the hyperbolic tangent construction is convenient, it is certainly possible to consider alternatives which can serve the same purpose.

One attractive alternative is to consider a rational polynomial. Since the level surface function $U(x, y)$ is a polynomial, this would represent the most pure method since it would keep the entire operation within the domain of polynomials. That is because a coefficient like $b$ appears linearly and thus so does $f(x)$. As a consequence, a multiplication by the polynomial denominator of the rational $f(x)$ will produce a new overall equivalent $U(x, y)$ that is entirely a polynomial. In mathematical terms, this means that we remain within the context of what are called algebraic varieties (a term from algebraic geometry).

With this motivation, we proceed. The conditions which must be met are that the blending function go from a constant value of $q$ along the negative $x$-axis to another constant value of $p$

along the positive $x$-axis. For this to happen asymptotically on either side of the origin, the highest power of $x$ in both the numerator and the denominator must match. The asymptotic value in this process is then the ratio of the coefficients for these respective highest powers. This, however, is just one number. But we actually need two different numbers if we wish to match two asymptotic values. Thus, this technique fails.

The only hope to retrieve it is to allow a large growth in negative values as we migrate along the axis in which there is to be no channel. For example, consider $b$ to be the rational function $x^3/(1+x^3)$. This has a singularity (pole) at $x = -1$. Departing the pole in the positive direction, the function increases from arbitrarily large negative values, passes through the origin, and continues along the positive $x$-axis to approach the asymptotic value of 1 from below. Departing the pole in the negative direction, the function decreases from arbitrarily large positive values and approaches the same asymptotic value from above. Thus, we must use the function on only the positive side of the pole. With the pole of b at -1, we shall consider a channel radius of 0.5 along the $y$-axis. This will be a safe distance away. We can then expect to have the wall on the negative side of the vertical channel to pass through the the $x$-axis ($y = 0$) somewhere between a value of the channel asymptote ($x = -0.5$) and the pole of $b$ (at $x = -1$). That offset distance will be determined by the choice of leading parameter $a$. To give a specific case, we will set $a = 1$. With this form, it is then natural to view the contour as a function $x$ of the variable $y$. This function is smooth and symmetric about $x = 0$. Upon evaluation at $y = 0$, the level surface function reduces to a fifth order equation in x which has an approximate solution of $x = -0.84$. By separating out $y$, an evaluation at the nearby $x = -0.8$ yields $y = \pm 0.92$. Thus we see the trend of going from the negative peak at -0.84 at $y = 0$ and dropping steadily to the asymptotic value of -0.5. Here, the offset distance is 0.34. In summary, we have witnessed that, while it is possible to use rational functions for the blending, more care and cleverness is required in its execution and the overall format is much more restrictive. In this paper, we will thus use the flexibility of transcendental functions so that we can conveniently employ multiple asymptotes in a single blending function. In particular, we shall utilize the hyperbolic tangent.

### Three dimensional models

Having established the basic properties in two dimensions, the extension into three dimensions can be discussed with more brevity. To begin, the 2D level surface function $U(x, y) = a - x^2y^2$ is replaced by the 3D generalization [5]

$$U(x, y, z) = 1 - (x^2y^2 + x^2z^2 + y^2z^2) \tag{5}$$

where now $\mathbf{r} = (x, y, z)$ and $r = |\mathbf{r}| = \sqrt{x^2 + y^2 + z^2}$. This gives the parallel to the previous 2D curves that decayed at the rate of $1/r$ as each axis was traversed to infinity. For the same reasons the one can be replaced by a parameter "$a$" to displace the effect of the decay. As before, this does not remove the decay. The correction to permit constant cross-sections is the same as before. This requires second order terms in each of $x, y$, and $z$ to asymptotically select the appropriate parts of the core quartic term. This gives us the basic function

$$U(\mathbf{r}) = a + bx^2 + cy^2 + dz^2 - (x^2y^2 + x^2z^2 + y^2z^2) \tag{6}$$

where $a, b, c$ and $d$ are constants.

With a simple analysis, we can show that for $a > -\min\{bc, bd, cd\}$, $b > 0$, Eq. 6 defines a surface of 6 tubes intersecting at origin. The separation in $x$ is explicitly seen by consideration of

the reordered function

$$U(\mathbf{r}) = x^2 \cdot (b - y^2 - z^2) + a + c \cdot y^2 + d \cdot z^2 - y^2 z^2 \tag{7}$$

As $\mathbf{r}$ departs the origin along the $x$-axis, the first term becomes more and more dominant over the others. In the level surface equation $U(\mathbf{r}) = 0$, a division by $x^2$ will give an equivalent equation where all terms except the first will approach 0 as $x$ grows in the outward vicinity of the $x$-axis. In the limit, the equation is just the circular cylinder of radius $\sqrt{b}$ about the $x$-axis.

The axes of the 6 tubes lay on the $\pm x$, $\pm y$ and $\pm z$ axes respectively. Asymptotically, the cross sections of the tubes are circles of radius $\sqrt{b}$, $\sqrt{c}$ and $\sqrt{d}$ for tubes along the $\pm x$, $\pm y$ and $\pm z$ axes respectively. The asymptotical error in radius is,

$$\Delta R < \frac{a + b + c + d}{2r^2} \tag{8}$$

The value of $a$ determines the abruptness of the intersection. A larger $a$ gives a lesser degree of abruptness of the intersection.

The signs of $b, c$ and $d$ determine whether tubes in the corresponding directions will present. If, say, $b$ is negative, the tubes in the $\pm x$ direction will disappear.

The analysis also shows that for the surface implicitly defined by $U(\mathbf{r}) = const$, it is well behaved near $const = 0$.

The function can be generalized to enable the tube branchs to be switched on and off individually:

$$U(\mathbf{r}) = a + Bx^2 + Cy^2 + Dz^2 - (x^2y^2 + x^2z^2 + y^2z^2) \tag{9}$$

where $B, C$ and $D$ are functions of $\mathbf{r}$. A set of choices are:

$$\begin{aligned} B &= -1 + s_x^+ \cdot (1 + \tanh(4x)) + s_x^- \cdot (1 - \tanh(4x)) \\ C &= -1 + s_y^+ \cdot (1 + \tanh(4y)) + s_y^- \cdot (1 - \tanh(4y)) \\ D &= -1 + s_z^+ \cdot (1 + \tanh(4z)) + s_z^- \cdot (1 - \tanh(4z)) \end{aligned} \tag{10}$$

Here, a particular choice of $B, C$ and $D$ is selected by setting six switch parameters to 0 or 1. Each of the parameters controls one tube branch and a value of 1 (value of 0) means there is (is not) a tube in the corresponding direction.

$$\begin{aligned} \mathbf{s_x^+} &= 0 \text{ or } 1 \text{ for tube along } +x \text{ axis.} & \mathbf{s_x^-} &= 0 \text{ or } 1 \text{ for tube along } -x \text{ axis.} \\ \mathbf{s_y^+} &= 0 \text{ or } 1 \text{ for tube along } +y \text{ axis.} & \mathbf{s_y^-} &= 0 \text{ or } 1 \text{ for tube along } -y \text{ axis.} \\ \mathbf{s_z^+} &= 0 \text{ or } 1 \text{ for tube along } +z \text{ axis.} & \mathbf{s_z^-} &= 0 \text{ or } 1 \text{ for tube along } -z \text{ axis.} \end{aligned} \tag{11}$$

The functional form can further generalized to allow the cross section of the tubes to be superellipses of different degrees. The further generalized formulas are listed in Appendix A.

## GRID GENERATION WITH GridPro/az3000

Now we show how one can generate grids for the region inside the branching tubes defined above using **GridPro/az3000**.

The grid generation process with **GridPro/az3000** starts with the design of a block topology (domain decomposition into hexahedras). Let's first consider the case where all 6 tube branches are present. The surface under consideration is

$$U(\mathbf{r}) = 2 + x^2 + y^2 + z^2 - (x^2y^2 + x^2z^2 + y^2z^2).$$

Here, all 6 tube branches have the same radius, 1. We also decide that the grid will be generated in the region $-5 < x < 5$, $-5 < y < 5$ and $-5 < z < 5$.

In order to have the flexibility of changing configuration easily, we will build the block topology in a component style. A component in **GridPro/az3000** represents a subtopology or a portion of the block topology that is conveniently grouped such that it can be reused in a similar fashion as a subroutine can in, say, Fortran.

How a block topology should be designed and how the various components should be chosen depend on many factors. One of our considerations here is to be able to change the configuration to include a different number of tubes rather quickly.

At this step, we choose to use two top level components, a `center` component for the center intersection and a `tube` for a tube branch. Each of these two components is constructed by one or more copies of the component, `sec`. The `sec` component, in turn, is constructed by properly linking two `loop4` components into a hyperquad.

These topology components are shown in Fig 2. Here, a solid dot is a block corner defined in the component, a solid line is a block edge defined in the component, and a circle is an imported corner into the component. On the corner and link level, the components `sec` and `tube` look the same. They differ mainly in whether the outer quad defines a block face. It is a face for `sec`, and it is not a face for `tube`.

To run **GridPro/az3000**, the topology design or components must be programmed in the Topology Input Language(TIL). The complete TIL code for this case is listed in Appendix B. In the following, we focus on three components to illustrate the general flavor of TIL programming.

The most basic component in the above design is `loop4`. In TIL code it looks like this :

**Program 1** *TIL component* `loop4`

```
COMPONENT loop4(sIN surf1,surf2, cIN pos[1..4],corn1[1..4],corn2[1..4])
BEGIN
 c 1 @<pos:1> -s surf1 surf2 -L corn1:1 corn2:1;
 c 2 @<pos:2> -s surf1 surf2 -L corn1:2 corn2:2 1;
 c 3 @<pos:3> -s surf1 surf2 -L corn1:3 corn2:3 2;
 c 4 @<pos:4> -s surf1 surf2 -L corn1:4 corn2:4 3 1;
END
```

This component imported two surfaces `surf1` and `surf2`, and three corner arrays of length 4, `pos[1..4]`, `corn1[1..4]` and `corn2[1..4]`. `surf1`, `surf2`, `pos`, `corn1` and `corn2` are similar to what are termed as dummy variables in Fortran subroutines. They do not introduce new surfaces or corner to the topology. Instead, they only provide a mechanism to refer to existing corners and surfaces defined outside of the component. Corners and surfaces are constructed with corner and surface definition statements in components, and by `INPUT`ing components that have corner and surface definition statements in them. In `loop4`, 4 corners are defined. For every call to this

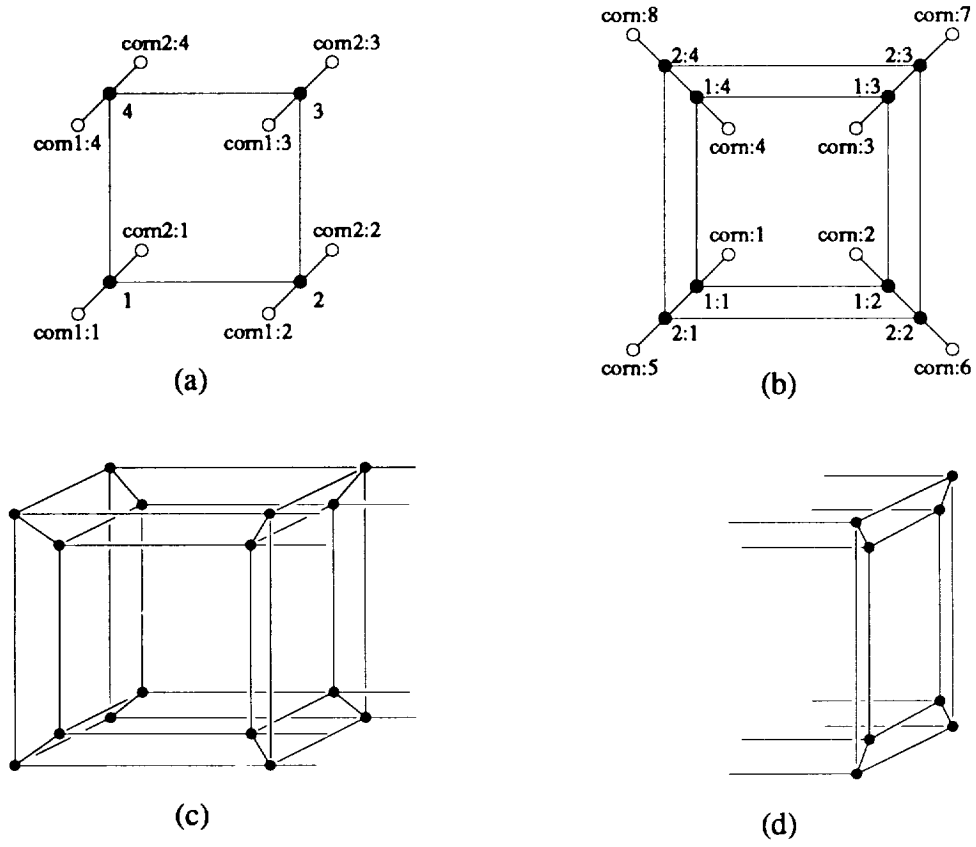



Figure 2: Topology components. (a). The most basic component, **loop4**. (b). The component, **sec**. It is constructed from two **loop4**'s. (c). The component, **center**. It is constructed from two **secs**. Also, 8 open links for connecting to a tube branch are shown. (d). The component, **tube**. It is constructed by a **sec** with certain boundary conditions. Shown are 8 open links for connecting to the **center**.

component, 4 corners will be constructed and inserted into the topology. A corner is generally defined by specifying a corner id, an initial position, a list of surfaces that the corner is supposed to be on, and a list of existing corners to which the current corner should have links. The surface list and link list are optional. In this case, the initial positions are provided by the import variable **pos**. All 4 corners are attached to both **surf1** and **surf2**. In addition, the corners are linked to the imported corners **corn1** and **corn2**. They are also properly linked to each other to form a 4-corner loop. The notation **corn1:3** simply means the $3^{rd}$ element of array **corn1**.

Two **loop4**'s are used to construct the component **sec**.

**Program 2** *TIL component* **sec**

```
COMPONENT sec(sIN tube,tube_end,cIN pos[1..4],corn[1..8])
BEGIN
  VECTOR p[1..4],shift;
  INPUT 1 loop4(sIN  (tube),(tube_end),
                cIN  (pos:1..4),(corn:1..4),(-4),
```

```
                cOUT (1..4));
  <shift> = 0.05*0.25*(<pos:1>+<pos:2>+<pos:3>+<pos:4>);
  <p:1> = 0.8*<pos:1>+0.2*<pos:3>-<shift>;
  <p:2> = 0.8*<pos:2>+0.2*<pos:4>-<shift>;
  <p:3> = 0.8*<pos:3>+0.2*<pos:1>-<shift>;
  <p:4> = 0.8*<pos:4>+0.2*<pos:2>-<shift>;
  INPUT 2 loop4(sIN  (tube_end),(-1),
                cIN  (p:1..4),(corn:5..8),(1:1..4),
                cOUT (1..4));
END
```

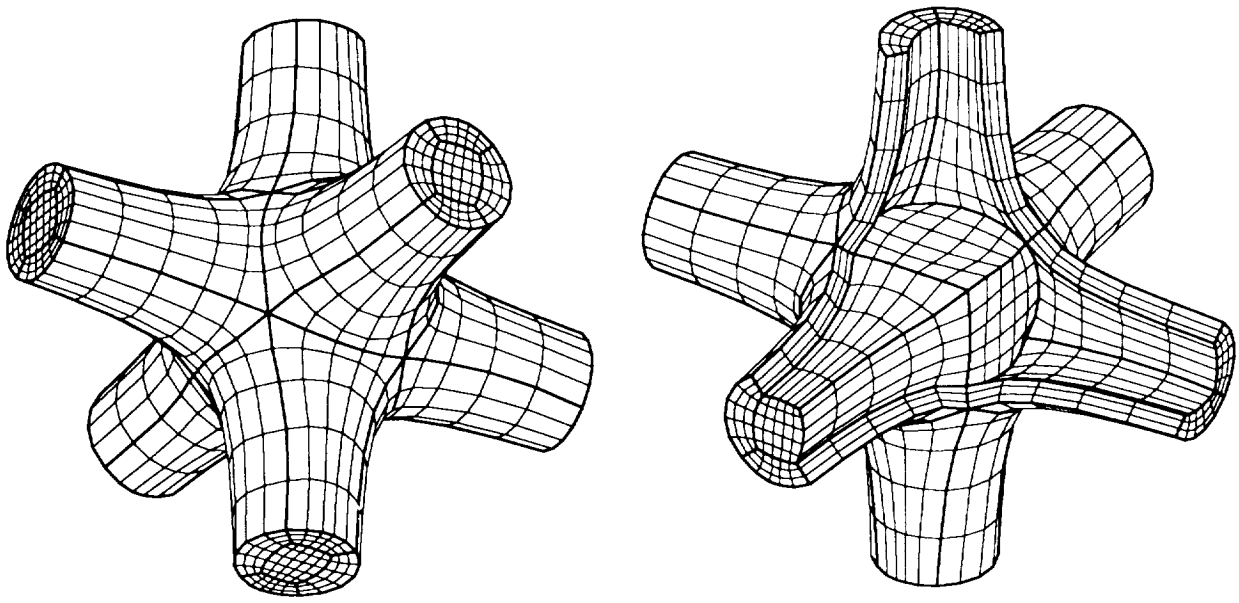
Here, vector operations are used to position the second `loop4` relative to the first. The key word `cOUT` exports some or all corners defined in the inputing component. Other components are similarly constructed.

To complete the topology design, we need to have a head component to assemble the various components together. This component must be the first one in the file. For our case, this component is named `Branching_Tubes`:

**Program 3** *TIL component* **`Branching_Tubes`**

```
COMPONENT Branching_Tubes()
BEGIN
 VECTOR cut_x,cut_X,cut_y,cut_Y,cut_z,cut_Z;

  s 1 -implic ''6jxXyYzZ.surf'' ; # 6 tubes
# s 1 -implic ''6jxXyYZ.surf'' ;  # 5 tubes
# s 1 -implic ''6jxXyY.surf'' ;   # 4 tubes
# s 1 -implic ''6jXYZ.surf'' ;    # 3 tubes

 <cut_x> = {-5, 0, 0}; <cut_X> = { 5, 0, 0};
 <cut_y> = { 0,-5, 0}; <cut_Y> = { 0, 5, 0};
 <cut_z> = { 0, 0,-5}; <cut_Z> = { 0, 0, 5};

  INPUT 1 center(sIN (1), cOUT (3:1..48));

  INPUT 2 tube_x(sIN (1),cIN (cut_x),(1:1..48));
  INPUT 3 tube_X(sIN (1),cIN (cut_X),(1:1..48));
  INPUT 4 tube_y(sIN (1),cIN (cut_y),(1:1..48));
  INPUT 5 tube_Y(sIN (1),cIN (cut_Y),(1:1..48));
  INPUT 6 tube_z(sIN (1),cIN (cut_z),(1:1..48));
  INPUT 7 tube_Z(sIN (1),cIN (cut_Z),(1:1..48));
END
```

Here a # symbol introduces comments ending at the end of the line. A TIL statement starting with the key word `s` defines a surface. Here, one surface is defined. It is an implicit type and the surface specification is in the file, 6jxXyYzZ.surf which contains the surface in Eq. 9..12 with all six



(a) (b)

Figure 3: A grid for the inside of a 6-tube intersection. The thicker lines are block boundaries. (a). an outside view. (b). a cut away view.

tube branches present. The file, 6jxXyYzZ.surf is in the C programming language macro definition style and is listed below.

**Program 4** *Surface File* **6jxXyYzZ.surf**

```
#define FUNCU (xa=x*x, x1= xa*tanh(x*4), \
               ya=y*y, y1= ya*tanh(y*4), \
               za=z*z, z1= za*tanh(z*4), \
2.0 + xa + ya + za - xa*ya - ya*za - za*xa)
```

For a non-build-in implicit surface, up to 9 pre-named macros must be defined. However, for a fixed surface such as the one we have here, only **FUNCU** is needed. **x**, **y** and **z** are the coordinate variables, and **x1**, **y1**, **z1**, **xa**, **ya** and **za** are intermediate variables. **x1**, **y1** and **z1** are not really used in the final formula. They will be used in a 5-tube surface. They are included here for the reason of easy comparison.

In Program 3, following the surface definition, a **center** component and 6 tube branches components are inputed. Six vectors are used to allow different choices of tube lengths. A high quality grid can be generated by running **GridPro/az3000** on this topology. Fig 3 shows some aspects of the generated grid. It consists of 31 elementary blocks. Grid densities, clustering and other aspects of grids can be readily adjusted by setting proper parameters.

Now, suppose we want to generate a grid for the same surface, but without the tube branch on the -z axis. The head component will be modified to look like this:

**Program 5** *Modified TIL component* **Branching_Tubes**

```
COMPONENT Branching_Tubes()
BEGIN
 VECTOR cut_x,cut_X,cut_y,cut_Y,cut_z,cut_Z;

# s 1 -implic ''6jxXyYzZ.surf'' ; # 6 tubes
  s 1 -implic ''6jxXyYZ.surf'' ;  # 5 tubes
# s 1 -implic ''6jxXyY.surf'' ;   # 4 tubes
# s 1 -implic ''6jXYZ.surf'' ;    # 3 tubes

 <cut_x> = {-5, 0, 0}; <cut_X> = { 5, 0, 0};
 <cut_y> = { 0,-5, 0}; <cut_Y> = { 0, 5, 0};
 <cut_z> = { 0, 0,-5}; <cut_Z> = { 0, 0, 5};

  INPUT 1 center(sIN (1), cOUT (3:1..48));

  INPUT 2 tube_x(sIN (1),cIN (cut_x),(1:1..48));
  INPUT 3 tube_X(sIN (1),cIN (cut_X),(1:1..48));
  INPUT 4 tube_y(sIN (1),cIN (cut_y),(1:1..48));
  INPUT 5 tube_Y(sIN (1),cIN (cut_Y),(1:1..48));
# INPUT 6 tube_z(sIN (1),cIN (cut_z),(1:1..48));
  INPUT 7 tube_Z(sIN (1),cIN (cut_Z),(1:1..48));
END
```

We used a different surface specification file "6jxXyYZ.surf", which has only 5 tube branches on $\pm x$, $\pm y$ and $+z$ axes respectively. To accomplish this in "6jxXyYZ.surf", the **FUNCU** macro is defined as:

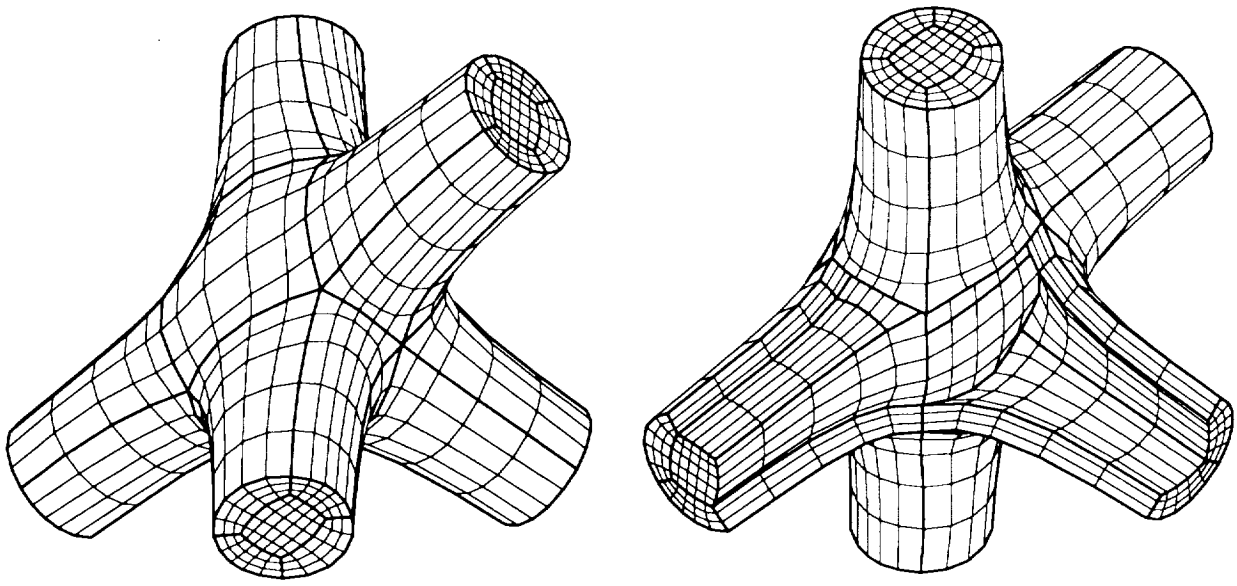
**Program 6** *Surface File* **6jxXyYZ.surf**

```
#define FUNCU (xa=x*x, x1= xa*tanh(x*4), \
               ya=y*y, y1= ya*tanh(y*4), \
               za=z*z, z1= za*tanh(z*4), \
        2.0 + xa + ya + z1 - xa*ya - ya*za - za*xa)
```

The second thing we did in Program 5 is to comment out **INPUT 6**, which builds the sub-topology for a tube branch on the $-z$ axis. Running **GridPro/az3000** on this topology yields the grid shown in Fig 4. Similarly, we can define a surface that has 4 tube branches or 3 tube branches, and appropriately modify the component **Branching_Tubes** for the corresponding topologies. The grids are shown in Fig. 5.

## CONCLUSION

We have now witnessed an application of implicit surfaces in the full cycle of geometry modeling and grid generation using **GridPro/az3000**. We have noted the high quality of both the geometry model and the generated grid therein. We have also noted the ease with which changes can be made



(a) (b)

Figure 4: A grid for the inside of a 5-tube intersection. The thicker lines are block boundaries. (a). an outside view. (b). a cut away view.

to both aspects. The example class of multi filleted tubes has demonstrated a general philosophy in a rather concrete setting which helps to establish a basic understanding.

In a forthcoming paper, we will develop a well-defined and easy-to-follow procedure for implicit surface modeling which allows the user to efficiently assemble simple implicit surfaces (e.g cylinders) into a complex whole (e.g intersecting cylinders). With these results, we have provided a powerful means to address many analysis applications, and moreover, have opened a path in the direction of rapid prototyping in grid generation.

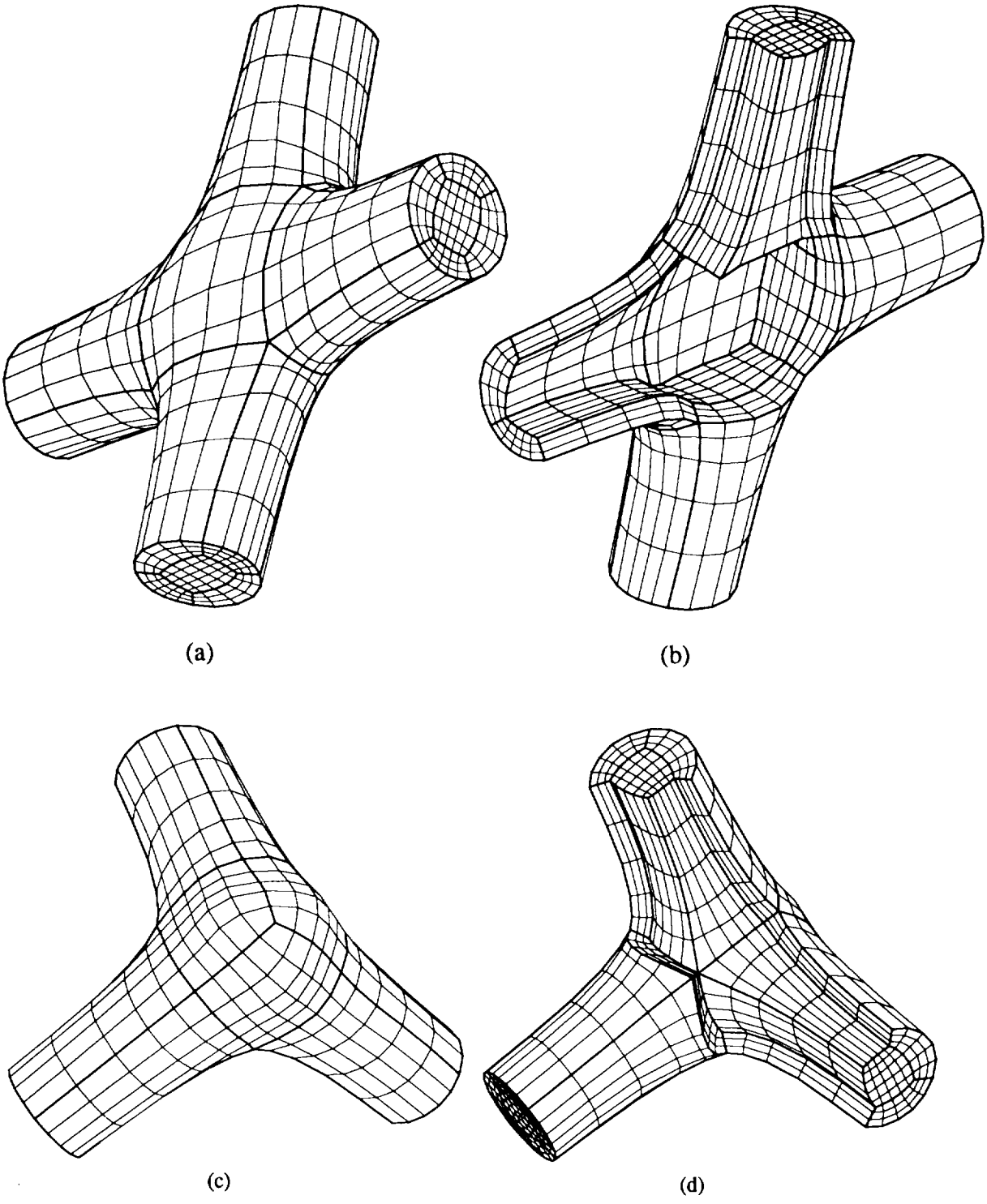



Figure 5: Grid for the inside of a 4-tube and 3 tube intersections. The thicker lines are block boundaries. (a). an outside view for the 4-tube case. (b). a cut away view for the 4-tube case. (c). an outside view for the 3-tube case. (d). a cut away view for the 3-tube case.

## APPENDIX A

To enlarge the scope of the geometry modeling for tube intersections, the function is generalized to allow the cross section of the tubes to be superellipses of different degrees that can be switched on and off individually. This generalization is given by,

$$U(\mathbf{r}) = A + BX^2 + CY^2 + DZ^2 - (X^2Y^2 + X^2Z^2 + Y^2Z^2)$$

where $A, B, C, D, X, Y$ and $Z$ are functions of $\mathbf{r}$.

$$X = |\frac{x}{S_x(\mathbf{r})}|^{n(\mathbf{r})} , \quad Y = |\frac{y}{S_y(\mathbf{r})}|^{n(\mathbf{r})} , \quad Z = |\frac{z}{S_z(\mathbf{r})}|^{n(\mathbf{r})} .$$

$$\begin{aligned} n(\mathbf{r}) = \frac{1}{r^2}\{ & [n_x^+ \cdot \frac{1+\tanh(4x)}{2} + n_x^- \cdot \frac{1-\tanh(4x)}{2}] \cdot x^2 \\ + & [n_y^+ \cdot \frac{1+\tanh(4y)}{2} + n_y^- \cdot \frac{1-\tanh(4y)}{2}] \cdot y^2 \\ + & [n_z^+ \cdot \frac{1+\tanh(4z)}{2} + n_z^- \cdot \frac{1-\tanh(4z)}{2}] \cdot z^2\} . \end{aligned}$$

$$\begin{aligned} S_x(\mathbf{r}) = \frac{1}{r^2}\{ x^2 & + [a_{xy}^+ \cdot \frac{1+\tanh(4y)}{2} + a_{xy}^- \cdot \frac{1-\tanh(4y)}{2}] \cdot y^2 \\ & + [a_{xz}^+ \cdot \frac{1+\tanh(4z)}{2} + a_{xz}^- \cdot \frac{1-\tanh(4z)}{2}] \cdot z^2\} . \end{aligned}$$

$$\begin{aligned} S_y(\mathbf{r}) = \frac{1}{r^2}\{ & [a_{yx}^+ \cdot \frac{1+\tanh(4x)}{2} + a_{yx}^- \cdot \frac{1-\tanh(4x)}{2}] \cdot x^2 + y^2 \\ + & [a_{yz}^+ \cdot \frac{1+\tanh(4z)}{2} + a_{yz}^- \cdot \frac{1-\tanh(4z)}{2}] \cdot z^2\} . \end{aligned}$$

$$\begin{aligned} S_z(\mathbf{r}) = \frac{1}{r^2}\{ & [a_{zx}^+ \cdot \frac{1+\tanh(4x)}{2} + a_{zx}^- \cdot \frac{1-\tanh(4x)}{2}] \cdot x^2 \\ + & [a_{zy}^+ \cdot \frac{1+\tanh(4y)}{2} + a_{zy}^- \cdot \frac{1-\tanh(4y)}{2}] \cdot y^2 + z^2\} . \end{aligned}$$

$$\begin{aligned} A &= constant(\text{say}, 1.2) \\ B &= -1 + s_x^+ \cdot (1+\tanh(4x)) + s_x^- \cdot (1-\tanh(4x)) \\ C &= -1 + s_y^+ \cdot (1+\tanh(4y)) + s_y^- \cdot (1-\tanh(4y)) \\ D &= -1 + s_z^+ \cdot (1+\tanh(4z)) + s_z^- \cdot (1-\tanh(4z)) \end{aligned}$$

These functions are controlled by a set of parameters. They are,

$\mathbf{s_x^+}$ = 0 or 1 for tube along $+x$ axis. $\mathbf{s_x^-}$ = 0 or 1 for tube along $-x$ axis.
$\mathbf{s_y^+}$ = 0 or 1 for tube along $+y$ axis. $\mathbf{s_y^-}$ = 0 or 1 for tube along $-y$ axis.
$\mathbf{s_z^+}$ = 0 or 1 for tube along $+z$ axis. $\mathbf{s_z^-}$ = 0 or 1 for tube along $-z$ axis.

$n_x^+$ - power of super ellipse for tube along $+x$ axis.
$n_x^-$ - power of super ellipse for tube along $-x$ axis.
$n_y^+$ - power of super ellipse for tube along $+y$ axis.
$n_y^-$ - power of super ellipse for tube along $-y$ axis.
$n_z^+$ - power of super ellipse for tube along $+z$ axis.
$n_z^-$ - power of super ellipse for tube along $-z$ axis.

$a_{xy}^+$ - length of $x$-semi axis for tube on $+y$ axis.
$a_{xy}^-$ - length of $x$-semi axis for tube on $-y$ axis.
$a_{yx}^+$ - length of $y$-semi axis for tube on $+x$ axis.
$a_{yx}^-$ - length of $y$-semi axis for tube on $-x$ axis.
$a_{zx}^+$ - length of $z$-semi axis for tube on $+x$ axis.
$a_{zx}^-$ - length of $z$-semi axis for tube on $-x$ axis.
$a_{xz}^+$ - length of $x$-semi axis for tube on $+z$ axis.
$a_{xz}^-$ - length of $x$-semi axis for tube on $-z$ axis.
$a_{yz}^+$ - length of $y$-semi axis for tube on $+z$ axis.
$a_{yz}^-$ - length of $y$-semi axis for tube on $-z$ axis.
$a_{zy}^+$ - length of $z$-semi axis for tube on $+y$ axis.
$a_{zy}^-$ - length of $z$-semi axis for tube on $-y$ axis.

## APPENDIX B

Files used to generate the grids with **GridPro/az3000**.

**Program 7** *Topology File* **6joint.fra**

```
SET GRIDDEN 6

COMPONENT Branching_Tubes()
BEGIN
 VECTOR shift_x,shift_X,shift_y,shift_Y,shift_z,shift_Z;

  s 1 -implic "6jxXyYzZ.surf" ; # 6 tubes
# s 1 -implic "6jxXyYZ.surf" ;  # 5 tubes
# s 1 -implic "6jxXyY.surf" ;   # 4 tubes
# s 1 -implic "6jXYZ.surf" ;    # 3 tubes

 <shift_x> = {-5, 0, 0}; <shift_X> = { 5, 0, 0};
 <shift_y> = { 0,-5, 0}; <shift_Y> = { 0, 5, 0};
 <shift_z> = { 0, 0,-5}; <shift_Z> = { 0, 0, 5};

 INPUT 1 center(sIN (1), cOUT (3:1..48));

 INPUT 2 tube_x(sIN (1),cIN (shift_x),(1:1..48));
 INPUT 3 tube_X(sIN (1),cIN (shift_X),(1:1..48));
 INPUT 4 tube_y(sIN (1),cIN (shift_y),(1:1..48));
 INPUT 5 tube_Y(sIN (1),cIN (shift_Y),(1:1..48));
 INPUT 6 tube_z(sIN (1),cIN (shift_z),(1:1..48));
 INPUT 7 tube_Z(sIN (1),cIN (shift_Z),(1:1..48));
END
```

```
# lines below are hidden from novice users
COMPONENT tube_x(sIN tube,cIN shift,corn[1..48])
BEGIN INPUT 1 tube(sIN (tube),cIN (shift),(corn:1..8)); END

COMPONENT tube_X(sIN tube,cIN shift,corn[1..48])
BEGIN INPUT 1 tube(sIN (tube),cIN (shift),(corn:9..16)); END

COMPONENT tube_y(sIN tube,cIN shift,corn[1..48])
BEGIN INPUT 1 tube(sIN (tube),cIN (shift),(corn:17..24)); END

COMPONENT tube_Y(sIN tube,cIN shift,corn[1..48])
BEGIN INPUT 1 tube(sIN (tube),cIN (shift),(corn:25..32)); END

COMPONENT tube_z(sIN tube,cIN shift,corn[1..48])
BEGIN INPUT 1 tube(sIN (tube),cIN (shift),(corn:33..40)); END

COMPONENT tube_Z(sIN tube,cIN shift,corn[1..48])
BEGIN INPUT 1 tube(sIN (tube),cIN (shift),(corn:41..48)); END

COMPONENT center(sIN tube)
BEGIN
  VECTOR pos[1..4],x,y,z;
  <x> = {1.5,0,0}; <y> = {0,1.5,0}; <z> = {0,0,1.5};
  <pos:1> =  <x> + <y> + <z>; <pos:2> = -<x> + <y> + <z>;
  <pos:3> = -<x> - <y> + <z>; <pos:4> =  <x> - <y> + <z>;
  INPUT 1 sec(sIN  (tube),(-1),
      cIN  (pos:1..4),(-8),     cOUT (1:1..4 2:1..4));
  <pos:1> =  <x> + <y> - <z>; <pos:2> = -<x> + <y> - <z>;
  <pos:3> = -<x> - <y> - <z>; <pos:4> =  <x> - <y> - <z>;
  INPUT 2 sec(sIN  (tube),(-1),
              cIN  (pos:1..4),(1:1..8), cOUT (1:1..4 2:1..4));

  INPUT 3 shuffle(cIN (1:2 2:2 2:3 1:3 1:6 2:6 2:7 1:7
       1:1 2:1 2:4 1:4 1:5 2:5 2:8 1:8
       1:4 1:3 2:3 2:4 1:8 1:7 2:7 2:8
       1:1 1:2 2:2 2:1 1:5 1:6 2:6 2:5
       2:1..8 1:1..8),
                  cOUT (corn:1..48),
       tube_x0(corn:1..8),  tube_x1(corn:9..16),
                       tube_y0(corn:17..24),tube_y1(corn:25..32),
                       tube_z0(corn:33..40),tube_z1(corn:41..48));
  g 1:1 1:5 3;
  LABEL SHELL = e(1:1 1:5);
END

COMPONENT shuffle(cIN corn[1..48]) BEGIN END

COMPONENT tube(sIN tube,cIN shift,corn[1..8])
```

```
BEGIN
  VECTOR pos[1..4],norm;
  <norm> = [-<shift>];

  s 1 -plane @(<norm>, <shift>);    # tube end

  <pos:1> = 0.7*(<corn:1> + <shift>); <pos:2> = 0.7*(<corn:2> + <shift>);
  <pos:3> = 0.7*(<corn:3> + <shift>); <pos:4> = 0.7*(<corn:4> + <shift>);
  INPUT 1 sec(sIN  (tube),(1),
      cIN  (pos:1..4),(corn:1..8), cOUT (1:1..4));
  x f 1:1 1:3 corn:1 corn:3;
END

COMPONENT sec(sIN tube,tube_end,cIN pos[1..4],corn[1..8])
BEGIN
  VECTOR p[1..4],shift;
  INPUT 1 loop4(sIN  (tube),(tube_end),
        cIN  (pos:1..4),(corn:1..4),(-4),
        cOUT (1..4));
  <shift> = 0.05*0.25*(<pos:1>+<pos:2>+<pos:3>+<pos:4>);
  <p:1> = 0.8*<pos:1>+0.2*<pos:3>-<shift>;
  <p:2> = 0.8*<pos:2>+0.2*<pos:4>-<shift>;
  <p:3> = 0.8*<pos:3>+0.2*<pos:1>-<shift>;
  <p:4> = 0.8*<pos:4>+0.2*<pos:2>-<shift>;
  INPUT 2 loop4(sIN  (tube_end),(-1),
cIN  (p:1..4),(corn:5..8),(1:1..4),
        cOUT (1..4));
END

COMPONENT loop4(sIN surf1,surf2,cIN pos[1..4],corn1[1..4],corn2[1..4])
BEGIN
 c 1 @<pos:1> -s surf1 surf2 -L corn1:1 corn2:1;
 c 2 @<pos:2> -s surf1 surf2 -L corn1:2 corn2:2 1;
 c 3 @<pos:3> -s surf1 surf2 -L corn1:3 corn2:3 2;
 c 4 @<pos:4> -s surf1 surf2 -L corn1:4 corn2:4 3 1;
END
```

**Program 8** *Schedule File* `6joint.sch`

```
step  1:  -S 100 -w

write -a -D 3 -f grid.tmp
```

**Program 9** *Surface File* `6jxXyYzZ.surf`

```
#define FUNCU (xa=x*x, x1= xa*tanh(x*4) \
               ya=y*y, y1= ya*tanh(y*4) \
```

```
              za=z*z, z1= za*tanh(z*4) \
2.0 + xa + ya + za   -xa*ya- ya*za -za*xa)
```

**Program 10** *Surface File* **6jxXyYZ.surf**

```
#define FUNCU (xa=x*x, x1= xa*tanh(x*4), \
               ya=y*y, y1= ya*tanh(y*4), \
               za=z*z, z1= za*tanh(z*4), \
2.0 + xa + ya + z1   -xa*ya- ya*za -za*xa)
```

**Program 11** *Surface File* **6jxXyY.surf**

```
#define FUNCU (xa=x*x, x1= xa*tanh(x*4), \
               ya=y*y, y1= ya*tanh(y*4), \
               za=z*z, z1= za*tanh(z*4),\
2.0 + xa + ya - za   -xa*ya- ya*za -za*xa)
```

**Program 12** *Surface File* **6jXYZ.surf**

```
#define FUNCU (xa=x*x, x1= xa*tanh(x*4), \
               ya=y*y, y1= ya*tanh(y*4), \
               za=z*z, z1= za*tanh(z*4), \
2.0 + x1 + y1 + z1   -xa*ya- ya*za -za*xa)
```

## References


[1] **GridPro**$^{TM}$**/az3000** is a software product of the Program Development Corp., 300 Hamilton Ave. Suite 409, White Plains, NY 10601.

[2] Bajaj C. L.; and Ihm I.: Algebraic Surface Design with Hermite Interpolation. *ACM Trans. on Graphics.*, Vol. 11. No. 1, Jan. 1992, pp. 61.

[3] Eiseman P .R.; and Cheng Z.; and Hauser J.: Applications of Multiblock Grid Generations with Automatic Zoning. *Proceedings of the* $4^{th}$ *International Conference* held at Swansea, Wales 1994, pp. 123.

[4] Program Development Corporation: **GridPro**$^{TM}$**/az3000** User's Guide and Reference Manual. 1993-1995.

[5] Gray A.: Modern Differential Geometry of Curves and Surfaces. published by CRC Press,Inc. 1993, pp. 226.

N95- 28773

# GridTool: A Surface Modeling and Grid Generation Tool

Jamshid Samareh-Abolhassani
Computer Sciences Corporation
NASA Langley Research Center
Hampton, Virginia

*GridTool is an interactive program for grid/geometry applications . Most grid generation programs represent geometry by a set of structured points which is not consistent with the Computer Aided Design (CAD) representation. The purpose of GridTool is to bridge the gap between the CAD and the grid generation systems.*

## Introduction

GridTool is designed around the concept that the surface grids are generated on a set of bi-linear patches. This type of grid generation is quite easy to implement, and it avoids the problems associated with complex CAD surface representations and associated surface parameterizations. However, the resulting surface grids are close to but not on the original CAD surfaces. This problem can be alleviated by projecting the resulting surface grids onto the original CAD surfaces. GridTool is designed primary for unstructured grid generation systems. Currently, GridTool supports VGRID [1] and FELISA [2] systems, and it can be easily extended to support other unstructured grid generation systems.

The data in GridTool is stored parametrically so that once the problem is set up, one can modify the surfaces and the entire set of points, curves and patches will be updated automatically. This is very useful in a multidisciplinary design and optimization process.

GridTool is written entirely in ANSI "C", the interface is based on the FORMS library [3], and the graphics is based on the GL library. The code has been tested successfully on IRIS workstations running IRIX4.0 and above. The memory is allocated dynamically, therefore, memory size will depend on the complexity of geometry/grid.

GridTool data structure is based on a link-list structure which allows the required memory to expand and contract dynamically according to the user's data size and action. Data structure contains several types of objects such as points, curves, patches, sources and surfaces. At any given time, there is always an active object which is drawn in magenta, or in their highlighted colors as defined by the resource file which will be discussed later.

## Surface Representation and Grid Projection

References [4]-[6] contain detailed descriptions of Surface Representation and Grid Projection. For completeness sake, a short summary will be provided here. In CAD systems, curves and surfaces are represented typically by NonUniform Rational B-Splines (NURBS) which is the most general mathematical representation for curves and surfaces. Most parametric curves and surfaces can be converted to an equivalent NURBS [7]-[8] representation without any loss of accuracy.

The surface-grid can be generated either in a parameter space or on approximated/simplified NURBS surfaces. Grids generated in a parameter space have two serious restrictions. The first restriction is that the choice of surface parameterization affects the CFD surface-grid. As shown in Refs. Ref. 6 and Ref. 9, a poor parameterization may cause the CFD surface-grid to be highly skewed. The second limitation is that a CFD surface-grid can not be generated over several overlapping NURBS surfaces which is the most serious restriction.

In the second method , the NURBS surfaces are approximated by a few bi-linear patches, then, the surface grids are generated based on these bi-linear patches. The resulting surface grids are close but they are not on the original NURBS surfaces. This problem can be alleviated by projecting the resulting grid points onto the original NURBS surfaces. This method is easy to implement, and it avoids the problems associated with surface parameterization. This method of grid generation will require a very robust and fast grid point projection.

## Advancing Front Applications (VGRID System)

In this section, using GridTool for VGRID system is described. VGRID system is a robust and fast unstructured grid generator developed by VIGYAN Inc. for NASA Langley Research Center. The VGRID code is fully functional and supported and can be obtained from NASA Langley Research Center (contact: Dr. Neal Frink, N.T.FRINK@LaRC.NASA.GOV). The VGRID system is based on an advancing front technique, and readers are referred to an excellent and detailed report by Parikh, Pirzadeh and Lö hner VGRID[1]. A short description of advancing front technique will be given here for the sake of completeness.

The advancing front method is an unstructured grid generation method similar to parabolic and hyperbolic methods for



PRECEDING PAGE BLANK NOT FILMED

structured grid generation. Grids are generated by marching from boundaries (front) towards the interior. First, the domain of interest is subdivided into a set of patches which cover the entire domain. Next, these patches are triangulated to form the "initial front". Finally, tetrahedral elements are generated based on the initial front. As tetrahedral elements are generated, the "initial front" is updated until the entire domain is covered with tetrahedral elements, and the front is emptied. The above process can be summarized in the following steps:

1. subdivide the domain of interest using GridTool,
2. specify grid spacings using GridTool,
3. generate the "initial front" using VGRID,
4. update the GridTool restart file to reflect the changes from VGRID using GridTool,
5. project the front onto the CAD surfaces using GridTool,
6. generate the volume grid using VGRID,
7. post–process the volume grid using VGRID.

The first step is to define the boundaries for the domain of interest. These boundaries are then subdivided into smaller patches using GridTool. In this paper, a patch is synonymous with a three–dimensional polygon. In the VGRID system, three types of patches are allowed: triangular Barnhill–Gregory–Nielson patches (three arbitrary sides), bilinear transfinite Coon's patch (four arbitrary sides), and planar patches (defined by an arbitrary number of sides, all lying in one plane). Each patch consists of several sides, and each side consists of several curves. In step 2, the grid spacing is defined by nodal and linear sources. An excellent description of these sources can be found in Ref. [10]. In step 3, all patches are triangulated to form the "initial front" using the VGRID system. In this step, VGRID may change the patch orientation. If so, in step 4, the GridTool restart file must be updated to reflect the changes. In step 5, the "initial front" is projected back onto the original surfaces using GridTool. In step 6, the volume grid is generated in one run or several restart runs using VGRID. In step 7, the volume grid can be post–processed to enhance grid quality. The details for steps 3, 5 and 6 can be found in Ref. 1.

## GridTool Interface

The interface consists of a main panel and several sub–panels. The panels consist of a set of buttons, input–fields, sliders, dials, positioners, browsers and message boxes. This section describes the user interaction with the GridTool interface. The user interacts with GridTool program by pointing/clicking the mouse buttons and the keyboard while the cursor is over a panel or the display window. Panels can be activated by pressing their buttons from the main window. They can be stowed away either from the panel itself by pressing the "*Stow*" button or by pressing the panel's button in the main panel. Whenever a panel is activated, the color of its button in the main panel will be changed from blue to green. Here is a list of actions and how they can be accomplished in the panels.

- To press a button, place the the cursor over the button and click with any of the three mouse buttons.
- To change the value in an input–field, place the the cursor over the input–field, click with any of the three mouse buttons, enter the value in the input–field, and complete the input by entering the "Return" key. The "ESC" key can be used to delete the entire field, or the "BackSpace" key can be used to delete a portion of it.
- To change the position of a slider, dial or a positioner, hold and drag the right mouse while the cursor is over the object.
- To select an object from a browse, place the the cursor over the object in the browser and click the right mouse. The background color of the selected object in the browser will change from black to green when they are selected.

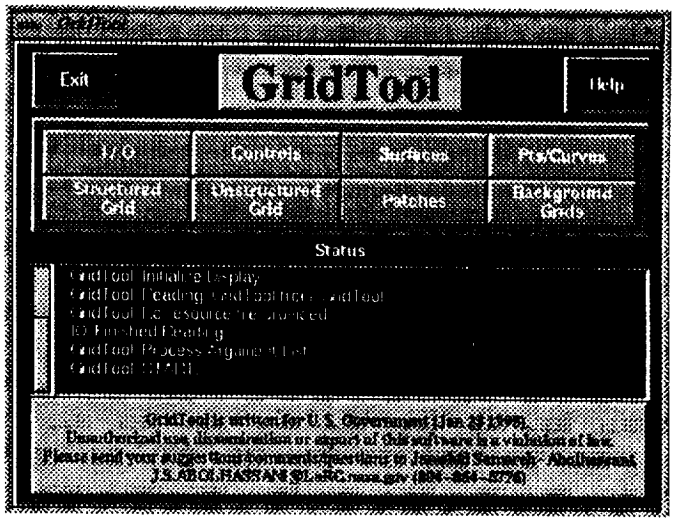



Fig. 1 Main Panel

ORIGINAL PAGE IS
OF POOR QUALITY

## Hot Keys

A series of hot keys are available in the display window which allow the user to accomplish some tasks without use of the panels. These keys can be activated by placing the cursor over the display window and clicking the hot key. The hot keys can be used to translate/rotate/zoom the object, to pick or to create an object.

---

## List of Hot Keys

| Keys | Action |
|---|---|
| Left Mouse | Translate (gridgen mode), rotate x, and y (PLOT3D mode) |
| Middle Mouse | Zoom, rotate z (PLOT3D) |
| Right Mouse | Translate (PLOT3D) |
| SHIFT + Mouse | Sparse Mode |
| b | Make a source active |
| c | Make a curve active |
| C | Move center of rotation to center of the active curve |
| f | Make a patch active |
| F | Move center of rotation to center of the active patch |
| g | Save the orientation |
| G | Restore the orientation |
| m | Move the active point to an existing point on a curve |
| n | Move the active source to an existing source and copy spacing |
| p | Make a point active |
| P | Move center of rotation to active point |
| r | Reset the image |
| s | Make a surface active |
| S | Move center of rotation to center of the active surface |
| t | Move the active point to an existing point on the active surface |
| w | Turn the active surface on/off |
| x | Turn axes on/off |
| F1 | The same as "*Next Curve*" button |
| F2 | The same as "*Next Point*" button |
| F5 | The same as "*Next Patch*" button |
| F6 | The same as "*Next Edge*" button |
| F7 | The same as "*Find Edge*" button |
| F8 | The same as "*Reverse the Active Patch*" button |
| F12 | Turn Surfaces on/off |

---

## GridTool Executions

GridTool can be executed by typing "GridTool" or "GridTool options filename", and here is a list of command line argument.

---

## Command Line Arguments

| Arguments | | Action |
|---|---|---|
| -h | | help |
| -f | restart_filename | read a restart file |
| -gf | gridgen_filename | read a gridgen formatted file |
| -g | gridgen_filename | read a gridgen binary file |
| -pf | plot3d_filename | read a plot3d formatted file |
| -p | plot3d_filename | read a plot3d binary file |
| -IGES | IGES_filename | read an IGES file |
| -felisa | | run GridTool in FELISA Mode |

---

When GridTool starts, it looks for the resource file, ".GridTool". This file could be at either the user's root, the current directory or defined by "setenv" unix command as "setenv GridTool_resources my_resource_filename". Users may change the resource file by customizing it to their needs. If the resource file does not exist, GridTool will use the defaults values which are listed in the Appendix A. The resource file may contain the preferred colors for displaying objects and the boundary conditions. A comment line can start either with a space or "#" in the first column. The default boundary conditions are based on the USM3D [11] code developed by Neal Frink at NASA/Langley Research Center. In the following sections, the application of GridTool for setting up data files for an advancing front technique (VGRID system)

is described. One important feature of GridTool is that each operation is accomplished in one step only. Therefore users can create and manipulate objects randomly.

There is some limited on–line help which can be activated by pressing the "*Help*" button in the main panel which in turn will open a browser. As the user moves the cursor over any object in the panels, a description of that button will be given in the browser.

## I/O

GridTool is capable of reading geometry/grid definitions in ASCII or C–Binary formats.

---

### File input/output formats

| File Type | Options |
|---|---|
| IGES [12] | read |
| RESTART | read/write |
| GRIDGEN [13] | read/write (Binary as well) |
| PLOT3D [14] | read/write (Binary as well) |
| CURVES | read/write |
| LaWGS [15] | read/write |
| VGRID-NET [1] | read/write |
| VGRID-FRONT [1] | read/write |
| VGRID-FRONT (Update) [1] | read/write |
| VGRID-d3m [1] | read/write |
| VGRID-d3m (Update) [1] | read/write |
| FELISA system [2] | read/write |

---

The IGES (Initial Graphics Exchange Standard) files are based on the industry standard as described in IGES manual [12]. GridTool is only capable of accepting the following entities: copious data (entity 106), lines (entity 110), parametric splines (entity 112), parametric surface spline (entity 114), NURBS curves (entity 126) and NURBS surfaces (entity 128). Surfaces defined by points can be read/written in GRIDGEN [13], PLOT3D [14], LaWGS [15] or VGRID–NET [1] formats.

The surface triangulation, "the initial front", can be read/written in a front format defined by the VGRID system [1] or FELISA system [2]. The necessary information for advancing front methods can be read/written either in a "d3m" input–file format for VGRID system [1] or in a "dat" input–file format FELISA system [2]. An ASCII "restart" file can be read/written at any time, which contains all created/modified/read objects. It is possible to combine several restart files to form one. This allows several people to work on the same configuration and combine all pieces at a later time. Before reading/writing a "d3m", "front" or a "dat" file, a "project name" must be selected. This name is used as the file name suffix for all necessary files (e.g. project.front). Once the file is read/written, the "*Files*" browser will be updated. To update the list displayed in "*Files*" browser, press the "*Update*" button.

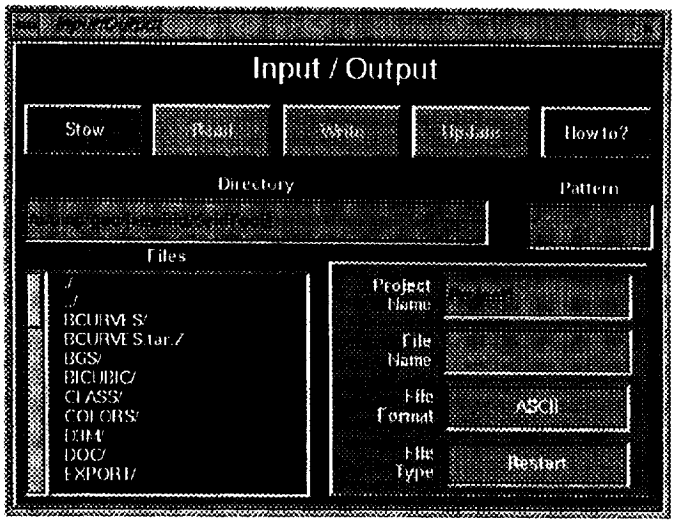



Fig. 2 I/O Panel

## Display and Viewing Controls

The display can be controlled either from the display window using the mouse and the keyboard, or from the "*Viewing Controls*" panel. The mouse can be used either in a default mode or in a PLOT3D [14] mode. The mode can be changed by pressing the "*PLOT3D*" button which is located in the "*Viewing Controls*" panel. The default mouse mode is similar to the GRIDGEN system [13]. In the default mode, while pressing the left mouse button, left, right, up, and down mouse movements will cause the object to translate in the corresponding directions. By holding the middle mouse down, up and down mouse movements will cause the object to zoom out and in. The object can be rotated using the numeric keypad, and this will be explained later. In the PLOT3D mode, by holding the left mouse down, left/right and up/down mouse movements will cause the object to rotate about the x and y screen coordinates, respectively. By holding the middle mouse down, left/right mouse movements will cause the object to rotate about the z screen coordinate, up/down mouse movements will cause the object to zoom out and in, respectively. By holding the right mouse down, left, right, up, and down mouse movements will cause the object to translate in the corresponding directions.

---

## Mouse Movements

| Mode | Movements | LM (down) | MM (down) | RM (down) |
|---|---|---|---|---|
| Default, | Right | Translate Right | | N/A |
| Default, | Left | Translate Left | | N/A |
| Default, | Up | Translate Up | Zoom out | N/A |
| Default, | Down | Translate Down | Zoom in | N/A |
| PLOT3D, | Right | Rotate Screen y | Rotate Screen z | Translate Right |
| PLOT3D, | Left | Rotate Screen -y | Rotate Screen -z | Translate Left |
| PLOT3D, | Up | Rotate Screen x | Zoom out | Translate Up |
| PLOT3D, | Down | Rotate Screen -x | Zoom in | Translate Down |

---

In either modes, the object can be rotated using the numeric keypad. The object can be rotated about two sets of axes: screen coordinates and body coordinates (world). The top row of the numeric keypad, the "Num Lock", "/" and "*" keys control the rotation about the x, y and z world coordinates, respectively. The second row, the "7", "8" and "9" keys control the rotation about the x, y and z screen coordinates, respectively. The rotation continues as long as the keys are pressed down. The object can be rotated ninety degrees by holding the "PageUp" or "PageDown" key while pressing the appropriate key on the numeric keypad. The object can be rotated in the reverse direction by holding down the "–" key from numeric keypad and the appropriate rotation keys. The object orientation can be reinitialized by pressing the "r" key which is the hot keys for resetting the object. All object manipulations can be accomplished from the "*Viewing Controls*" panel as well.

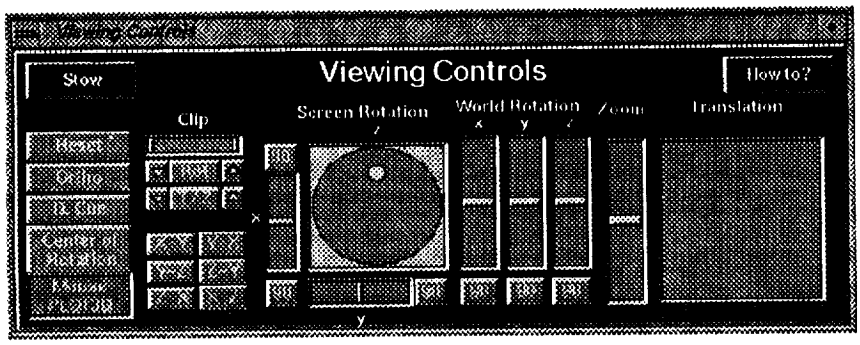



Fig. 3 Viewing Controls Panel

## Center of Rotation

This panel is designed to allow the user to move the center of rotation to an arbitrary point in space. The center of rotation can be moved to: an exiting point by using the three sliders, "*x*", " y", "z" to centers of the active surface/curve/patch/point by pressing the appropriate buttons. The center of rotation can be reset by pressing the "*Reset CR*" button. The hot keys, P, C, F, S can be used in the display window to move the center of rotation to the active point, active curve, active patch and active surface.

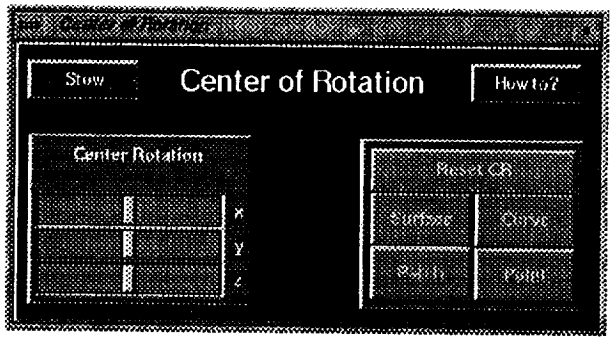



Fig. 4 Center of Rotation Panel

## Surface Attributes

This panel is designed to allow the user to manipulate the surface properties. In order to change the background color of the display window, the "*Background*" button in the "*Attributes*" panel should be pressed. Then, the color can be changed by moving three sliders for the colors or by inputing the RGB color number (Red, Green, Blue) in the input–fields.

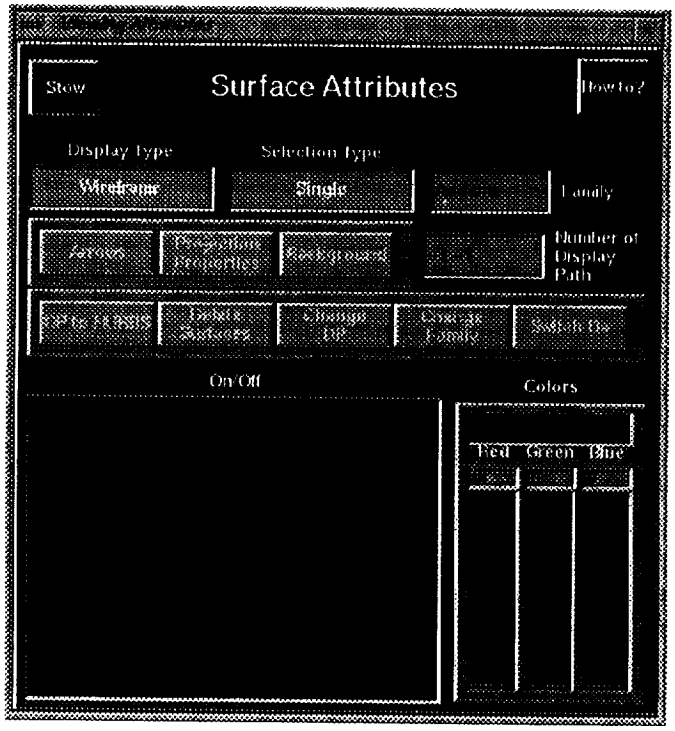



Fig. 5 Surface Attributes Panel

The surface properties such as color, direction and number of display paths can be changed in this panel. First, the "*Surfaces*" button in this panel should be pressed, and then "*On/Off*" browser will appear. In order to change the properties of some of the surfaces, first they must be turned on. This can be done either from the display window or from the "*On/Off*" browser by placing the cursor over the surface number and clicking with the left mouse. In order to turn a surface on/off from the display window, first the surface must become an active surface by placing the cursor over the surface and hitting the hot key "s". Then, the active surface can be turned on/off by hitting the hot key "w". Once the appropriate surfaces are turned on, then the properties can be changed. Surface colors can be changed by using the three sliders for colors or by inputing the RGB color number in the input–fields. The direction of surfaces can be changed by pressing the "*Directions*" button. The number of display paths can be changed by inputing the desired numbers of display paths in u and v coordinates in the input–field for "*number of display path*" and then pressing "*Change DP*" button.

## Projection Properties

This panel is designed for modifying the projection parameters, and it displays the projection parameters for the active surface. For detail description of these properties, readers are referred to Refs. [4]–[6]. In this panel, it is also possible to change the parameters such that part of a surface is displayed and projected to. This can be done by changing the limits

of the surface parameters, us (minimum u), ue (maximum u), vs (minimum v), and ve (maximum v). The "*Itmax*" is the number of iteration for the projection to converge. The "*EPS*" is the residual limit for the projection to converge. The "*du*" and "*dv*" are the step size in u and v directions, respectively. The "*magn*" are parameters used to pull grid points to boundaries, and their values range from 0–1.

1. "*mag1*" for minimum v
2. "*mag2*" for maximum u
3. "*mag3*" for maximum v
4. "*mag4*" for minimum u

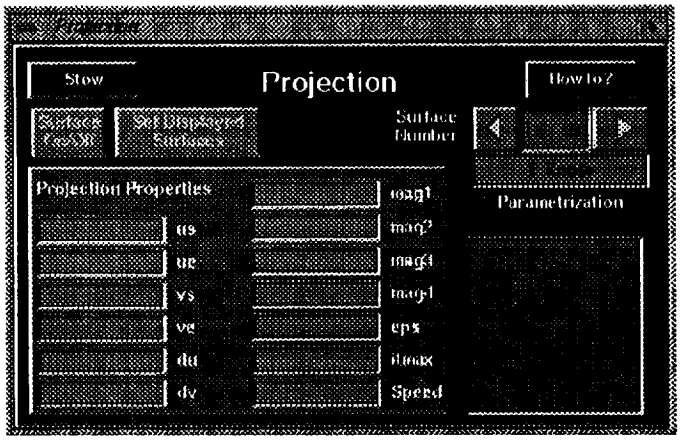



**Fig. 6 Projection Properties Panel**

## Points/Curves

This panel contains several buttons, input–fields and a positioner, and they are used to create/modify/delete points and curves. In GridTool, a curve is represented by a link–list of points. These points are either on a surface (surface points) or somewhere in space (XYZ points). For surface points, in addition to the x, y, z, the surface number and its parametric coordinates, u and v are stored in the data base. Curves can be created together as a family. For example, all curves associated with a wing could be created together as a "wing" family. The family name for curves is selected from the "*Patches*" panel which will be discussed later. To start a new curve, press the "*Next Curve*" button. To create a new point for a curve, the "*Next Point*" button should be pressed, and this newly created point becomes the active point. Since every operation is done in one step, the new point will be placed where the last point was, and then the user can move the new point, (the active point), to any location. The active point can be moved to an existing point on a curve by placing the cursor over the desired curve point and hitting the hot key "m". Similarly, the active point can be moved to an existing point on the active surface by placing the cursor over the surface point and hitting the hot key "t". Also, the active point can be moved to any location on the active surface by either: (1) typing the parametric coordinates in the "*U & V*" input box, (2) moving the "*U & V*" positioner, (3) moving the "*U & V*" sliders. In order to move the active point in space, first the point should be converted to an "XYZ" point by pressing "*On Surface*" button. Once the active point is an XYZ point, the "*x*", "*y*", and "*z*" sliders can be used to move the point to anywhere in the space. It is also possible to change the coordinate by typing the values in the input field boxes. A point can be inserted ahead of the active point on a curve by pressing the "*Insert Point*" button. The new point becomes the active, and if the two neighboring points are on the same surface, then the inserted point will also be on the same surface. The point or the active curve can be deleted by pressing the "*Delete Point*" or the "*Delete Curve*" buttons, respectively. A point or a curve can become active by placing the cursor over it and hitting the hot key "p" or "c", respectively.

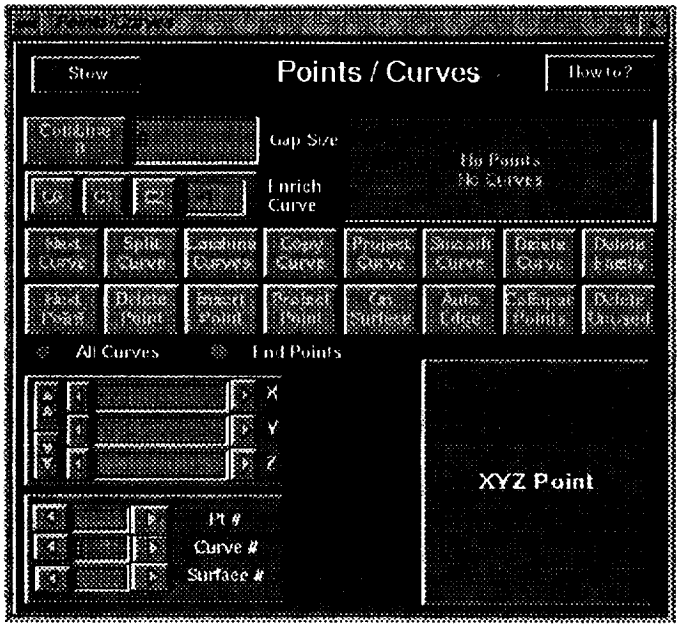



Fig. 7 Points/Curves Panel

## Patches

A patch is a closed three–dimensional polygon which is defined by a set of curves. Nonplanar patches should be 3– or 4–sided, and planar patches could be n–sided. Each side of a patch could consist of several curves. Each patch is stored as a link–list of curves. To create a patch, the "*Next Patch*" button should be pressed. Then, the first curve for the patch must be activated by the user, and then it can be accepted as the first curve by pressing "*Accept Edge*" button. The subsequent curves can be added by letting GridTool find them. This can be done by pressing "*Find Edge*" button until the correct curve is found. GridTool will find the next curve within the tolerance define in the " *tol*" input–field box. Once the correct curve is found, it should be accepted by pressing the "*Accept Edge*" button. Once a patch is created, its direction can be reversed by pressing the "*Reverse Patch*" button.

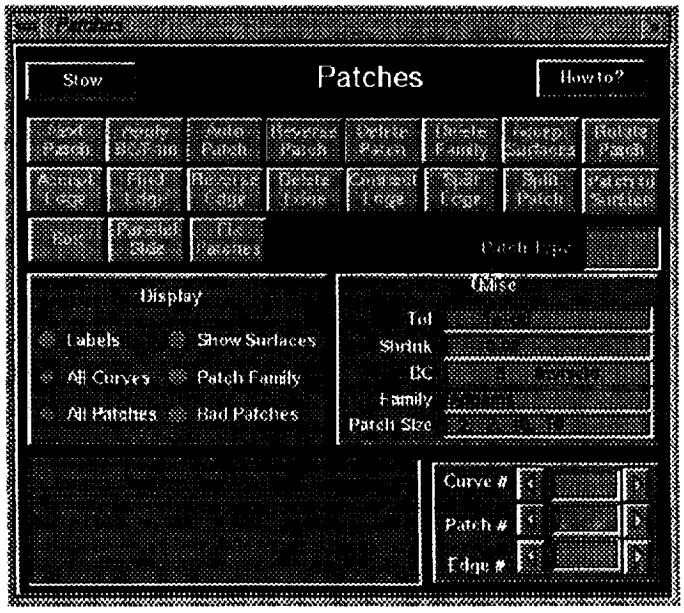



Fig. 9 Patches Panel

## BOX

This panel is designed to allow the user to create/delete a box. To create a box, press "*Create Box*" button which will create a box bounded by minmax in x, y, z coordinates defined in the six input–fields in the panel. The resulting curves

and patches are grouped together as the "Box" family. To delete the box, press the "*Delete Box*" button which will delete all patches and curves in the "Box" family.

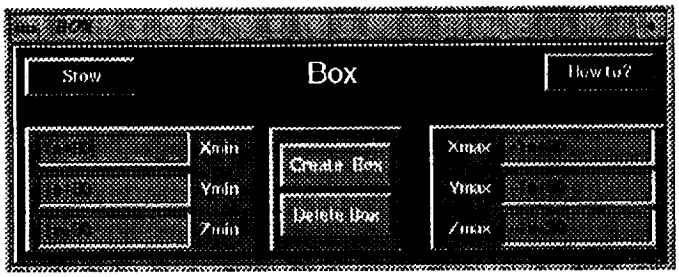



Fig. 10 BOX Panel

## Bg Grid

This panel contains several buttons and input–fields, and they are used to create/modify/delete nodal and linear sources. In order to define grid spacing, nodal and linear sources must be created and placed in the right locations. In order to create a source, the "*Bg Grid*" and "*Points/Curves*" panels must be activated. To create a source, press the "*Next Source*" button. This will create a source similar to the last source. If this is the first source, it will create a nodal source and place it in the middle of the domain. The location of a source can be moved by using the same techniques as described for moving points. The value spacings, "S1" and "S2", are the sizes of ideal tetrahedrals for the source locations. An excellent description of parameters "a_n, b_n, alpha" can be found in Ref. 10.

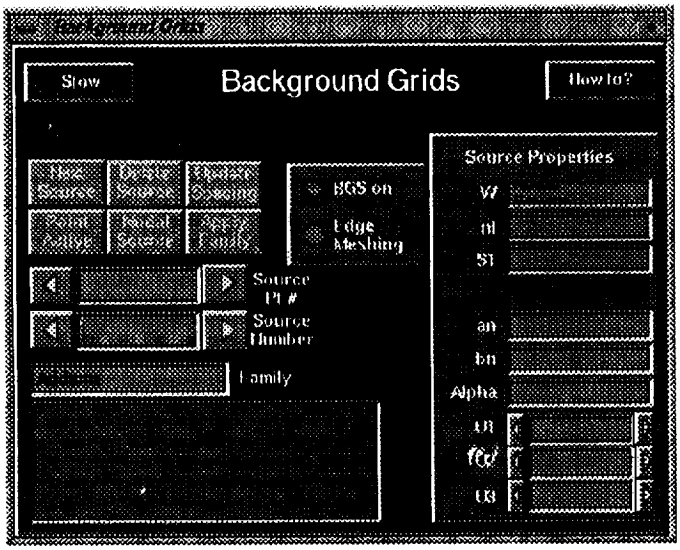



Fig. 11 Background Grid Panel

## Unstructured Grid

This panel is designed to manipulate the unstructured surface grid. The "*Front*" button is used to activate the "*Front*" from which a surface triangulation can be projected onto a set of surfaces.

## Front

This panel is used to project the surface triangulation (front) onto a set of surfaces, and this process can be divided into five steps: (1) read the front using the "*IO*" panel, (2) turn the appropriate patches on, (3) turn associated surfaces on, (4) project the front onto the surfaces, and (5) check for the validity of the new triangulation. In order to have a successful projection, users are required to insure that: (1) the surface triangulation is close enough to the associated surfaces, and (2) the associated surfaces have sufficient display paths.

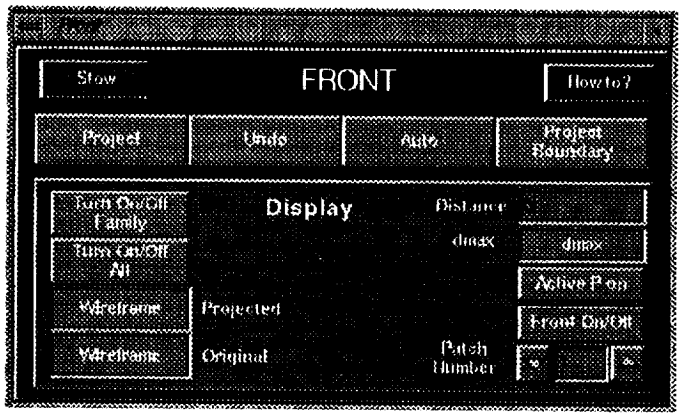



Fig. 12 Unstructured Grid Panel

## Structured Grid

This panel is used to project the entire or portion of a structured surface grid onto a set of surfaces, and this process can be divided into five steps: (1) read the surface definition and the surface grid using the "*IO*" panel, (2) turn on associated surfaces, (3) make the surface to be projected active, (4) project the surface grid onto the surfaces, and (5) check for the validity of the new surface grid. In order to have a successful projection, users are required to insure that: (1) the surface grid is close enough to the associated surfaces, and (2) the associated surfaces have sufficient display paths. User may undo the projection or reinterpolate a protion of the surface grid.

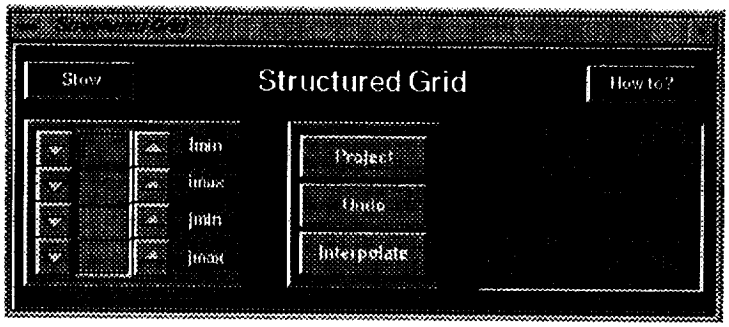



Fig. 13 Structured Grid Panel

## Acknowledgments

I would like to thank Mr. Javier Garriz for his assistance and invaluable suggestions. I would also like to thank Neal Frink for his support, and the following people for their invaluable comments and suggestions: Steve Alter, Brent Bates, Norma Bean, Will Bene, Mike Bockelie, Kamran Fouladi, Clyde Gumbert, Bill Jones, Victor Lessard, Wendy Lessard, Naomi McMillin, Paresh Parikh, Shahryar Pirzadeh Ram Shan, John Stewart, James Weilmuenster, Mike Wiese.

## References

1. Parikh, P. Pirzadeh, S., and Lö hner, T., "*A Package for 3-D Unstructured Grid Generation Finite-Element Flow Solution and Flow Field Visualization,*" NASA CR-182090, September 1990.
2. Peraire, J., Peiro, J., Formaggia, L., Morgan, K., Zienkiewics, O., " *Finite Element Euler Computations in Three Dimensions,*" International Journal for Numerical Methods in Engineering, vol 26, pp. 2135-2159, 1988.
3. Overmars, Mark H., " *A Graphical User Interface Toolkit for Silicon Graphics Workstations,* Department of Computer Sciences, Utrecht University, the Netherlands, 1992.
4. Samareh-Abolhassani, Jamshid, *"Unstructured Grid on NURBS Surfaces.",* AIAA-Paper 93-3454.
5. Samareh-Abolhassani, Jamshid, "*Unstructured Surface Grid Generation,*" NASA CP-10119, pp. 223-251, September 1993.
6. Samareh-Abolhassani, Jamshid, *"Triangulation of NURBS Surfaces,"* in Numerical Grid Generation in Computational Fluid Dynamics and Related Fields, ed. Weatherill, N. P., et. al., Pineridge Press, pp. 377-388, 1994.
7. Tiller, Wayne, *"Theory and Implementation of Non-Uniform Rational B-Spline Curves and Surfaces: class notes for NASA Langley,"* August 1991.
8. Farin, G. E., *Curves and Surfaces for Computer Aided Geometric Design, A Practical Guide,"* Second Edition, Academic Press, New York, 1989.

9. Samareh-Abolhassani, Jamshid, Stewart, John E., *"Surface Grid Generation in a Parameter Space,}"* AIAA-92-2717, 1992.
10. Pirzadeh, S.,'' *Structured Background Grids for Generation of Unstructured Grids by Advancing Front Method,* " AIAA-91-3233, 1991.
11. Frink, N. T., '' *Three-Dimensional Upwind Schemes for Solving Euler Equations on Unstructured Grids,*" Ph. D. Dissertation, VPI, September 1991.
12. '' *The Initial Graphics Exchange Specification (IGES) Version 5.0,*" Distributed by National Computer Association, Administrator, IGES/PDES Organization, 2722 Merrilee Drive, Suite 200, Fairfax,VA 22031.
13. Steinbrenner, John P., Chawner, John R., Fouts, Chris L., '' *The GRIDGEN 3D Multiple Block Grid Generation System,*" Contract Report F33615-87-C-3003, General Dynamics, July 1990.
14. Buning, Pieter, '' *Plot3d User Guide,* ", NASA Ames, 1990.
15. Anonymous, '' *LaWGS: A Description of the Langley Wireframe Geometry Standard Format,*" NASA Technical Memorandum, 85767, February 1985.

## Appendix A: Sample Resource File

```
#
# This is a comment line
  This is also a comment line

# plot3d mouse movement
plot3d

#color  item                    R       G       B

#       Background Color for the Display
color   background              0       0       0       black

#       Color for the Active Surface
color   active_surface          199     21      133     medium violet red

#       Colors for the Points
color   xyz_pt                  67      110     238     royal blue
color   active_pt               255     0       0       red
color   surface_pt              34      139     34      forest green

#       Color for the Curves
color   xyz_curve               100     149     237     cornflower blue
color   active_curve            255     0       0       red
color   surface_curve           34      139     34      forest green

#       Color for the Patches
color   normal_patch            219     112     147     Pale violet red
color   active_edge             199     21      133     medium violet red
color   active_patch            199     21      133     medium violet red

#       Colors for the Background Grid
color   normal_bgs              255     255     0       yellow
color   active_bgs              255     0       0       red
color   active_bgs_pt           199     21      133     medium violet red
#
# BOUNDARY CONDITIONS Based on USM3D
#
bc      freestream              0
bc      reflection plane        1
bc      extrapolation           2
bc      inflow/outflow          3
bc      viscous                 4
bc      inviscid                5
bc      nacelle inlet           101
bc      nacelle exit            102
bc      inlet mass              110
bc      inlet pressure          111
bc      inlet mach              112
bc      inlet velocity          113
bc      special bc1             1001
bc      special bc2             1002
bc      special bc3             1003
bc      special bc4             1004
bc      special bc5             1005
```

N95-28774

# INTERACTIVE HIGH QUALITY GRID GENERATION

Ning Lu and Peter R. Eiseman
Program Development Corporation
300 Hamilton Avenue, Suite 409
White Plains, New York 10601

## ABSTRACT

We generate high quality grids interactively on a computer workstation through optimization for some simple yet commonly encountered geometric shapes.

## 1. INTRODUCTION

Algebraic and elliptic grid generation are the two most widely used methods for generating grids. Algebraic grid generation is known for fast interactive grid generation. The grid smoothness and lack of skewness, however, are usually not assured. On the other hand, elliptic grid generation can generate smooth grids but the time it takes to generate them is usually too long for interactive response. It is naturally desirable to combine the advantages of both methods and to generate high quality grids (ref. 1) fast and efficiently on a computer workstation. This requires people to strike a proper balance between the grid generation speed and the grid quality. We have pursued this idea for some simple yet commonly encountered geometric shapes. Many of them have a symmetry that can be used. We have been able to obtain high quality grids fast and interactively. Our method is to analyze each case carefully and to take full advantage of the region symmetry (if any) so that optimization is needed only for a part of geometric region under consideration. We have combined various grid generation methods, such as a fast elliptic grid generation technique (combining the control point form of algebraic grid generation with elliptic grid generation, ref. 2) and a transfinite interpolation with orthogonal boundary terms, to achieve an optimization for each individual case. After such an optimization, we have been able to generate excellent single- and multi-block grids quickly. This achievement enables us to generate certain single- and multi-block grids interactively with reasonable response even on modestly powered workstations.



PRECEDING PAGE BLANK NOT FILMED

## 2. GRID WITHIN A RHIOMBUS

A simple transfinite interpolation (TFI) will not generate an orthogonal grid in general. A more general TFI with boundary derivative terms can improve grid orthogonality. In certain cases, grid orthogonality can be achieved along edges of a face grid or on surfaces of a volume grid. This, in general, requires a careful analysis for each geometry region encountered. In this section, we present an example of carrying out such a detailed analysis.

We consider a surface grid bounded within a quadrilateral. Since each of four boundary edges is a straight line, a bicubically blended interpolation formula (i.e., with derivative information) can be used,

$$\mathbf{r}(u,v) = \underline{B}^T(u)\ \underline{S}\ \underline{B}(v)\ , \tag{1}$$

where $\underline{S}$ is a $4 \times 4$ matrix

$$\underline{S} = \begin{pmatrix} \mathbf{r}(0,0) & \mathbf{r}(0,1) & \mathbf{r}_v(0,0) & \mathbf{r}_v(0,1) \\ \mathbf{r}(1,0) & \mathbf{r}(1,1) & \mathbf{r}_v(1,0) & \mathbf{r}_v(1,1) \\ \mathbf{r}_u(0,0) & \mathbf{r}_u(0,1) & \mathbf{r}_{uv}(0,0) & \mathbf{r}_{uv}(0,1) \\ \mathbf{r}_u(1,0) & \mathbf{r}_u(1,1) & \mathbf{r}_{uv}(1,0) & \mathbf{r}_{uv}(1,1) \end{pmatrix}\ , \tag{2}$$

and $\underline{B}$ is a 4-component vector

$$\underline{B}(u) = \begin{pmatrix} B_1(u) \\ B_2(u) \\ B_3(u) \\ B_4(u) \end{pmatrix} = \begin{pmatrix} 1 - 3u^2 + 2u^3 \\ 3u^2 - 2u^3 \\ u - 2u^2 + u^3 \\ -u^2 + u^3 \end{pmatrix}\ . \tag{3}$$

In Eq. (1), $u$ and $v$ are parametric variables. The parametric region of $0 \le u \le 1$, $0 \le v \le 1$ corresponds to the area within the quadrilateral. In Eq. (2), $\mathbf{r}(0,0)$, $\mathbf{r}(1,0)$, $\mathbf{r}(0,1)$, and $\mathbf{r}(1,1)$ are four corners of the quadrilateral (i.e., the surface grid), four $\mathbf{r}_u$'s are partial derivatives with respect to the variable $u$ at four corners, four $\mathbf{r}_v$'s are partial derivatives with respect to the variable $v$ at four corners, and $\mathbf{r}_{uv}$'s are second-order mixed derivatives (also called twists) at four corners. In Eq. (3), the blending functions $B_j(u)$ have the properties: $B_1(u) + B_2(u) = 1$, $B_j(0) = \delta_{j1}$, $B_j(1) = \delta_{j2}$, $dB_j(0)/du = \delta_{j3}$, $dB_j(1)/du = \delta_{j4}$.

In grid generation, the positions of four corners $\mathbf{r}(0,0)$, $\mathbf{r}(1,0)$, $\mathbf{r}(0,1)$, and $\mathbf{r}(1,1)$ are known. However, the values of 8 first-order derivatives and 4 second-order derivatives are not given. With a proper choice of these derivatives, one can generate a better grid than those generated by the simple TFI or, for a quadrilateral, by the simple bilinear interpolation. The idea is to require that the grid direction at the boundary edges point to a desired direction. For example, on the $v = 0$ edge, the grid direction going inside the quadrilateral can be easily found from Eqs. (1)-(3) to be

$$\mathbf{r}_v(u,0) = B_1(u)\mathbf{r}_v(0,0) + B_2(u)\mathbf{r}_v(1,0) + B_3(u)\mathbf{r}_{uv}(0,0) + B_4(u)\mathbf{r}_{uv}(1,0)\ . \tag{4}$$

By requiring it to be perpendicular to the $v = 0$ edge (a straight line here), it is possible to achieve boundary orthogonality at certain places on the $v = 0$ boundary edge.

In the following, we take a rhombus as an example to illustrate how a detailed analysis can dramatically improve the grid orthogonality. A rhombus is a special parallelogram with opposite sides and opposite angles equal but having no right angle. For a rhombus centered at origin (on the $z = 0$ plane), its four corner positions can be written as (see fig. 1)

$$\mathbf{r}(1,1) = -\mathbf{r}(0,0) = h\,\mathbf{j}\,, \quad \mathbf{r}(1,0) = -\mathbf{r}(0,1) = w\,\mathbf{i}\,. \tag{5}$$

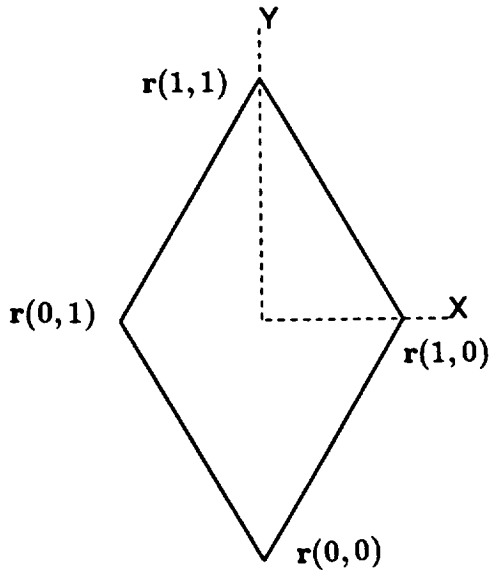



Figure 1.—A rhombus centered at origin.

From symmetry considerations, we know (i) that the magnitudes of $\mathbf{r}_u(0,0)$, $\mathbf{r}_v(0,0)$, $\mathbf{r}_u(1,1)$, and $\mathbf{r}_v(1,1)$ must be equal, and (ii) that the magnitudes of $\mathbf{r}_u(1,0)$, $\mathbf{r}_v(1,0)$, $\mathbf{r}_u(0,1)$, and $\mathbf{r}_v(0,1)$ must be equal too. In other words, we have the following relations:

$$\mathbf{r}_u(0,0) = \mathbf{r}_u(1,1) = \alpha\hat{\mathbf{u}}\,, \tag{6a}$$

$$\mathbf{r}_v(0,0) = \mathbf{r}_v(1,1) = \alpha\hat{\mathbf{v}}\,, \tag{6b}$$

$$\mathbf{r}_u(1,0) = \mathbf{r}_u(0,1) = \beta\hat{\mathbf{u}}\,, \tag{6c}$$

$$\mathbf{r}_v(1,0) = \mathbf{r}_v(0,1) = \beta\hat{\mathbf{v}}\,, \tag{6d}$$

where

$$\hat{\mathbf{u}} = (w\mathbf{i} + h\mathbf{j})/\sqrt{w^2 + h^2} \ , \quad \hat{\mathbf{v}} = (-w\mathbf{i} + h\mathbf{j})/\sqrt{w^2 + h^2} \tag{7}$$

are the unit vectors along the directions of $\mathbf{r}(1,0) - \mathbf{r}(0,0)$ and of $\mathbf{r}(0,1) - \mathbf{r}(0,0)$, respectively. It is clear that both $\alpha$ and $\beta$ are positive: $\alpha > 0$ and $\beta > 0$. Also, because of symmetry, (i) twists $\mathbf{r}(0,0)$ and $\mathbf{r}(1,1)$ do not have $x$ component whereas twists $\mathbf{r}(1,0)$ and $\mathbf{r}(0,1)$ do not have $y$ component, and (ii) the magnitudes of the twists at each pair of opposite corners are equal but point to opposite directions. Namely, twists satisfy the relations

$$\mathbf{r}_{uv}(1,1) = -\mathbf{r}_{uv}(0,0) = a\,\mathbf{j} \ , \tag{8a}$$

$$\mathbf{r}_{uv}(1,0) = -\mathbf{r}_{uv}(0,1) = b\,\mathbf{i} \ . \tag{8b}$$

Expressions (6) and (8) are the most general form allowed (by the symmetry requirements) for a rhombus. In other words, the symmetry consideration reduces the number of independent first-order derivatives from 8 to 2, and also reduces the number of independent second-order derivatives from 4 to 2. Now we only need to decide 4 scalar quantities: $\alpha$, $\beta$, $a$, and $b$. For the rhombus shown in fig. 1, Eq. (4) reduces to

$$\mathbf{r}_v(u,0) = [\alpha B_1(u) + \beta B_2(u)]\hat{\mathbf{v}} - aB_3(u)\,\mathbf{j} + bB_4(u)\,\mathbf{i} \ . \tag{9}$$

Without loss of generality, we assume in the following discussion that $h > w$. When $h > w$, the angle between the two unit vectors $\hat{\mathbf{u}}$ and $\hat{\mathbf{v}}$, which is the internal angle at corner $\mathbf{r}(0,0)$, is less than $90°$, since $\hat{\mathbf{v}} \cdot \hat{\mathbf{u}} = (h^2 - w^2)/(h^2 + w^2) > 0$. To determine the 4 quantities $\alpha$, $\beta$, $a$, and $b$, we first consider an (unnormalized) orthogonality function $f(u) \equiv \mathbf{r}_v(u,0)\cdot\hat{\mathbf{u}}$, which is a function (third-order polynomial) of the parametric variable $u$. Its value at the two ends $\mathbf{r}(0,0)$ and $\mathbf{r}(1,0)$ of the $v = 0$ edge is positive: $f(0) = \alpha\hat{\mathbf{v}} \cdot \hat{\mathbf{u}} > 0$ and $f(1) = \beta\hat{\mathbf{v}} \cdot \hat{\mathbf{u}} > 0$. We want to force the grid to be orthogonal at the middle of the $v = 0$ edge (but not overly doing this enforcement). Specifically, we require that the orthogonality function $f(u)$ reaches its minimum value at $u = \frac{1}{2}$ and that the minimum value of the orthogonality function is zero (for the interval $0 \le u \le 1$). These two requirements lead to two conditions for the orthogonality function $f(u)$,

$$\frac{df(\frac{1}{2})}{du} = 0 \ , \quad f(\tfrac{1}{2}) = 0 \ . \tag{10}$$

From these two conditions, we can express the twists $a$ and $b$ in terms of the first-order derivatives $\alpha$ and $\beta$,

$$a = \frac{(5\alpha - \beta)(h^2 - w^2)}{h\sqrt{w^2 + h^2}} \ , \tag{11a}$$

$$b = \frac{(5\beta - \alpha)(h^2 - w^2)}{w\sqrt{w^2 + h^2}} \ . \tag{11b}$$

Eliminating the twists $a$ and $b$ in the orthogonality function $f(u)$ with the help of Eqs. (11), $f(u)$ becomes

$$f(u) = \frac{1}{2}\hat{\mathbf{v}} \cdot \hat{\mathbf{u}} \left[(\alpha+\beta)(2u-1)^2 - (\alpha-\beta)(2u-1)^3\right] , \tag{12}$$

which indeed reaches its minimum value of 0 at $u = \frac{1}{2}$ (for the interval $0 \leq u \leq 1$) for any positive $\alpha$ and $\beta$.

Next, we ask $\mathbf{r}_v$ and $\mathbf{r}_u$ to be orthogonal at the center of the rhombus $\mathbf{r}(\frac{1}{2},\frac{1}{2}) = 0$. Due to symmetry, the angle between $\mathbf{r}_v(\frac{1}{2},\frac{1}{2})$ and $\mathbf{j}$ is always equal to the angle between $\mathbf{r}_u(\frac{1}{2},\frac{1}{2})$ and $\mathbf{j}$. Thus, asking $\mathbf{r}_v$ and $\mathbf{r}_u$ to be orthogonal at the center of the rhombus is equivalent to requiring that $\mathbf{r}_v(\frac{1}{2},\frac{1}{2}) \cdot \mathbf{j} = \mathbf{r}_v(\frac{1}{2},\frac{1}{2}) \cdot (-\mathbf{i})$ and

$$\mathbf{r}_u(\tfrac{1}{2},\tfrac{1}{2}) \cdot \mathbf{i} = \mathbf{r}_u(\tfrac{1}{2},\tfrac{1}{2}) \cdot \mathbf{j} . \tag{13}$$

Equation (13) leads to a third relation among the derivatives $\alpha$, $\beta$, $a$, and $b$,

$$\mathbf{r}_u(\tfrac{1}{2},\tfrac{1}{2}) \cdot (\mathbf{j}-\mathbf{i}) = \frac{h-w}{4\sqrt{w^2+h^2}}(6\sqrt{w^2+h^2} - \alpha - \beta) + \frac{a+b}{16} + \frac{3(\beta-\alpha)(w+h)}{8\sqrt{w^2+h^2}} . \tag{14}$$

Substituting Eqs. (11) into Eq. (14), we can eliminate $a$ and $b$ and obtain an expression relating $\alpha$ to $\beta$,

$$\alpha = \frac{(w+5h)\beta + 24wh(h-w)(w^2+h^2)^{-1/2}}{5w+h} . \tag{15}$$

Now we can still choose the value of $\beta$ as it fits. There is, however, a restriction. In the case of $h > w$ under discussion, the twists $a$ and $b$, as determined by Eqs. (11), should not be negative. Otherwise, the resulting grid will not look nice.

In practice, we find that a vanishing twist at the pair of obtuse angles of the rhombus, i.e.,

$$b = 0 , \tag{16}$$

is a good choice. This leads to

$$\alpha = 5\beta \tag{17}$$

according to Eq. (11b). It follows from Eqs. (15) and (17) that

$$\beta = \frac{h(h-w)}{\sqrt{w^2+h^2}} , \quad h > w . \tag{18}$$

Substituting Eqs. (17) and (18) into Eq. (11a), we find that correspondingly

$$a = \frac{24(h-w)(h^2-w^2)}{w^2+h^2} . \tag{19}$$

Equations (16)–(19) thus uniquely determine all 12 derivatives in Eq. (1).

Figure 2 shows three rhombuses inside a regular hexagon of side length $2w$. Two rhombuses with a pair of edges parallel to the $x$-axis can be obtained from the third rhombus (oriented as in fig. 1) with a simple rotation of $\pm 120°$ around the corner $\mathbf{r}(1,0)$ on $z = 0$ plane. For each rhombus in fig. 2, $h = \sqrt{3}w$ and Eqs. (18) and (19) give $\beta = 0.634w$ and $a = 8.785w$.

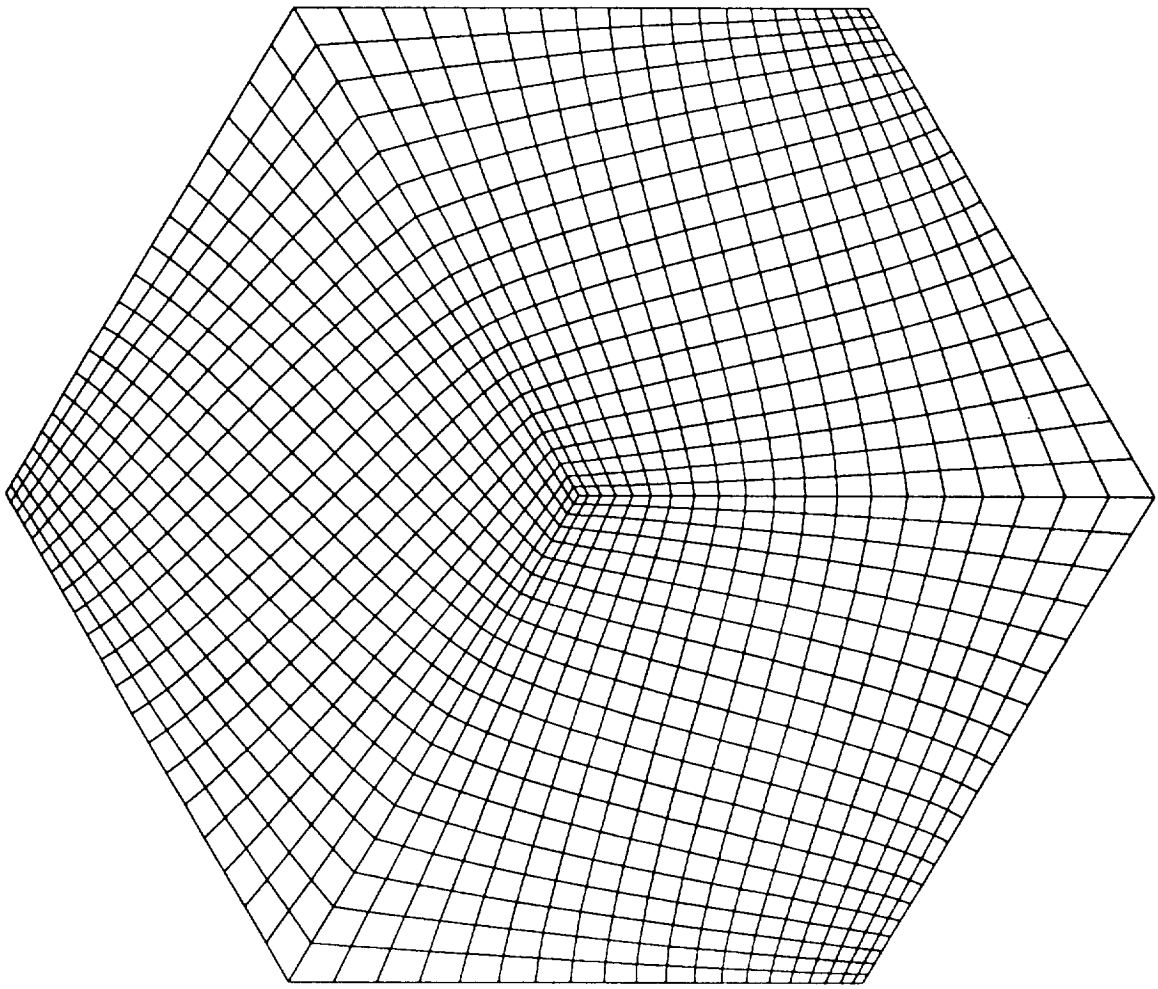


Figure 2.—A three-block grid inside a regular hexagon.

## 3. OTHER EXAMPLES

When the geometric region for a surface grid is no longer a quadrilateral (as in most cases), a TFI formula with boundary information terms, i.e., a bicubically blended TFI (see, for example, ref. 3), should be used to replace Eq. (1). By a careful analysis for each geometric region or shape under study, one can properly choose the directions and

magnitudes of boundary derivative terms and thus dramatically improve the grid quality and smoothness. In this way, a surface grid with much improved grid orthogonality can be generated. For a volume grid, one can also include boundary derivative terms into the often-used volume grid FTI formula and thus generate a volume grid with improved grid orthogonality. We have carried out this approach for some simple yet commonly encountered geometric shapes besides the rhombus case discussed above in section 2. Besides this pure algebraic grid generation approach, we have also combined the control point form of algebraic grid generation with elliptic grid generation (ref. 2). Here we first elliptically generate a sparse net of control points and then use the control point form of algebraic grid generation to generate a much denser grid. As a result, we generate a smooth grid with less computational time and storage space. This approach also enables us to obtain a high quality grid in an interactive environment on a computer workstation. We are incorporating these new grid generation techniques into a software product **GridPro**$^{TM}$/IGgen from Program Development Corporation, so that a user will be able to readily generate high quality grids interactively on a computer workstation. Currently, a collection of boundary geometries are provided and a user can choose among them. In addition, the user can also specify the grid type wanted, such as O-type, C-type, H-type, and/or multi-block grids. The user can also specify the grid density and clustering pattern. Once the user makes a choice, our software immediately generates high quality grids. In this section we show some of the grids generated interactively using **GridPro**/IGgen.

Figure 3 shows a single-block 2D grid for a passage which has a circular extra space. Due to symmetry, we only need to carry out grid orthogonality optimization for a half, a quarter, or even one eighth of the grid.

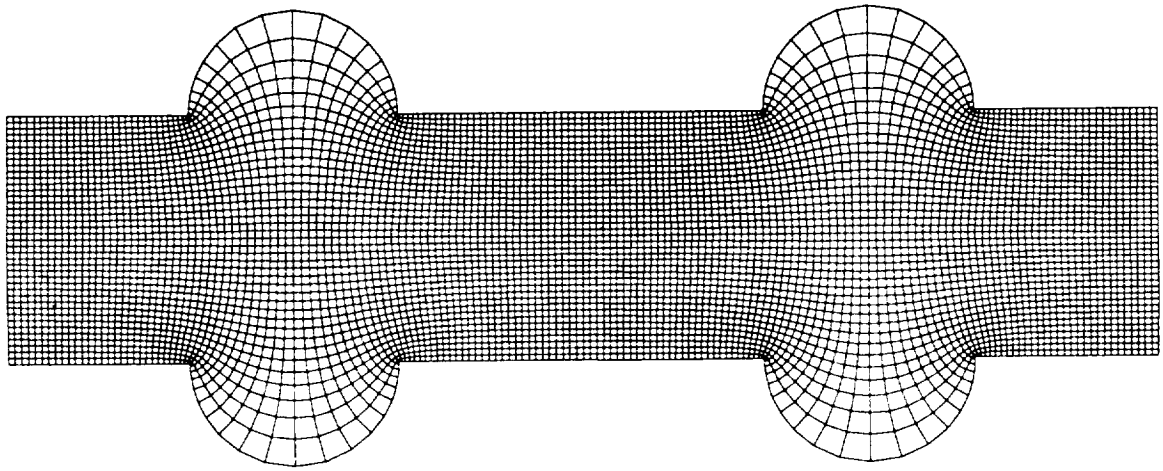


Figure 3.—A single-block 2D grid inside a 2D pipe which has symmetric circular extra space.

Figure 4 displays an H-type 2D grid passing around a cylinder whose cross section looks like a super circle. There are two blocks. One block is the mirror image of the other block about the horizontal symmetry line. For each block, there exists a vertical symmetry line. So we only need to do optimization for one quarter of the whole grid.

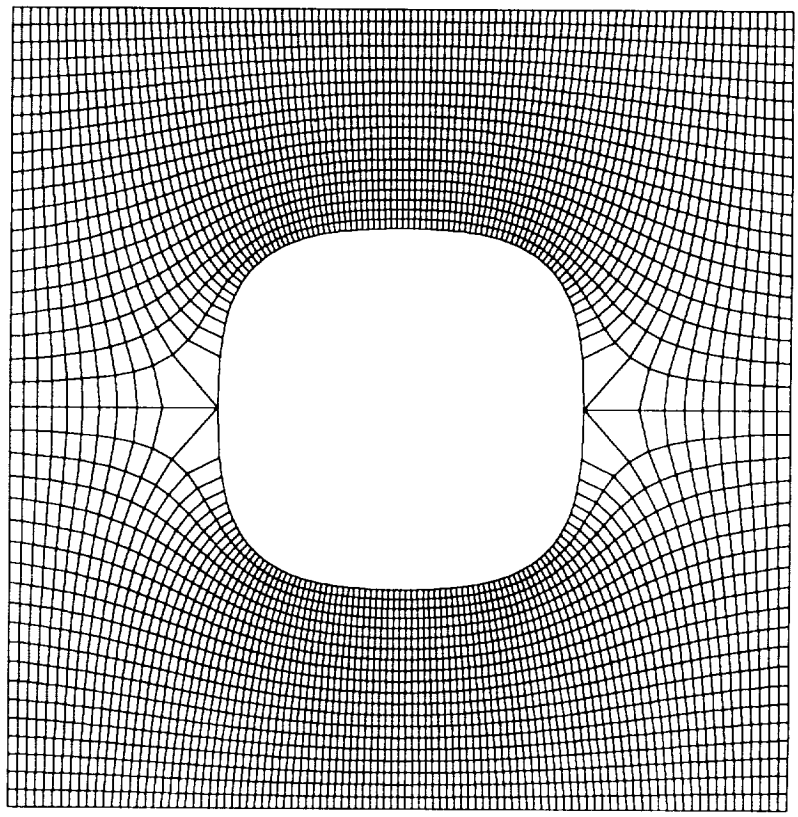


Figure 4.—A 2-block grid passing around a supercircle-like object.

Figure 5 shows a single-block volume grid inside one half of a torus. Five surfaces (except the top surface) are displayed. Symmetry of the geometry shape enables us to focus on one sheet of the volume grid and do grid orthogonality well. Other sheets can be obtained by rotations of different angles.

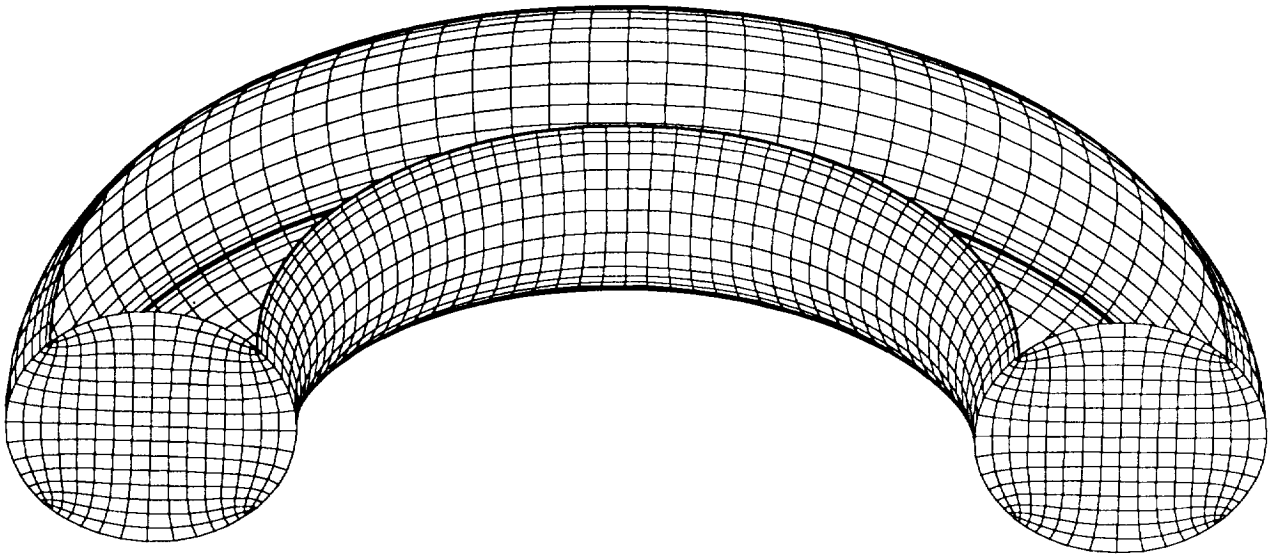


Figure 5.—Half of a single-block volume grid inside a torus.

Figure 6 shows two pictures of a 7-block volume grid inside a sphere. Because of symmetry, we only need to perform optimization for one of six side blcks and the center block.

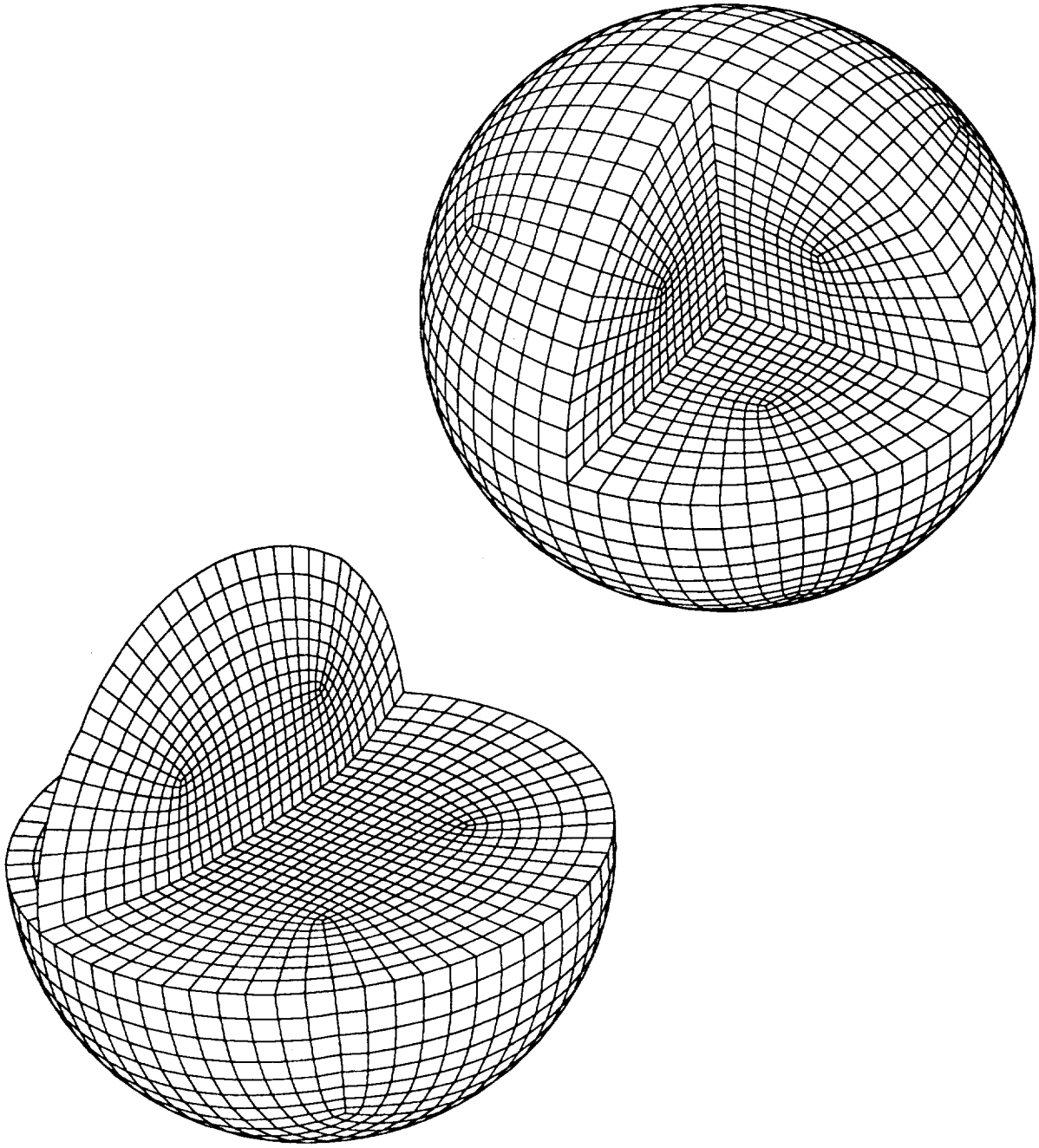


Figure 6.—Two views of a 7-block volume grid inside a sphere.

These examples clearly show the smoothness and lack of grid skewness that can be reliably expected in our new interactive environment.

## 4. SUMMARY

We have combined several new grid generation methods to obtain high quality grids in an interactive environment. Grids can be generated reliably, and grid quality is high in general. Currently, we are developing a grid generation program called **GridPro**$^{TM}$/IGgen to take advantage of these new advances in grid generation. Our goal is to generate a range of high-quality single- and/or multi-block grids interactively on a computer workstation.

## REFERENCES


1. Lu, N. and Eiseman, P.R., "A grid quality manipulation system," in *Numerical Grid Generation in Computational Fluid Dynamics and Related Fields*, A. S.-Arcilla, J. Häuser, P. R. Eiseman, and J. F. Thompson, eds. (North-Holland, Amsterdam, 1991), pp. 607–616.
2. Eiseman, P.R., Lu, N., Jiang, M., and Thompson, J.F., "Algebraic-Elliptic Grid Generation," in *Numerical Grid Generation in Computational Fluid Dynamics and Related Fields*, Weatherill, N.P., Eiseman, P.R., Häuser, J., and Thompson, J.F., eds. (Pineridge Press, Swansea, U.K., 1994), pp. 37–48.
3. G. Farin, "Curves and Surfaces for Computer Aided Geometric Design," 2nd ed. (Academic Press, Boston, 1990), Chapter 20.

N95- 28775

# NASA-IGES TRANSLATOR AND VIEWER

Jin J. Chou and Michael A. Logan
Computer Sciences Corporation
NASA Ames Research Center
Moffett Field, California 94035

## SUMMARY

NASA-IGES Translator (NIGEStranslator) is a batch program that translates a general IGES (Initial Graphics Exchange Specification) file to a NASA-IGES-Nurbs-Only (NINO) file. IGES is the most popular geometry exchange standard among Computer Aided Geometric Design (CAD) systems. NINO format is a subset of IGES, implementing the simple and yet the most popular NURBS (Non-Uniform Rational B-Splines) representation. NIGEStranslator converts a complex IGES file to the simpler NINO file to simplify the tasks of CFD grid generation for models in CAD format.

The NASA-IGES Viewer (NIGESview) is an Open-Inventor-based, highly interactive viewer/editor for NINO files. Geometry in the IGES files can be viewed, copied, transformed, deleted, and inquired. Users can use NIGEStranslator to translate IGES files from CAD systems to NINO files. The geometry then can be examined with NIGESview. Extraneous geometries can be interactively removed, and the cleaned model can be written to an IGES file, ready to be used in grid generation.

## INTRODUCTION

In April 1994 the NASA Geometry Data Exchange Specification for Computational Fluid Dynamics (NASA-IGES) was published (ref. 1). The intend of this specification is to facilitate geometry data exchange between CAD and Computational Fluid Dynamics (CFD) grid generation, and to provide high quality data to CFD processes. From the inception of the NASA-IGES standard, the Geometry Data Exchange Subcommittee (ref. 2) realize that it is very difficult to influent CAD system vendors to conform to a standard from a small user community. Hence, the best approach to handle the interface between CFD and CAD systems is to conform to the current CAD convention and yet to simplify the interface. Two steps were taken to achieve to this goal.

First, the IGES file format (ref. 3), the most popular data exchange format among CAD systems, was picked as the base of the CFD geometry standard. However, IGES contains hundreds of data exchange elements, called entities. It would be extremely onerous to CFD processes if they have to handle all the entities. Instead, the Subcommittee concentrated on entities that are essentials to CFD processes. Popular NURBS representation is chosen as the backbone of the standard; a few structural and annotational entities in IGES are also picked to facilitate, but not to complicate, the data exchange. A complete list of entities in the NASA-IGES standard can be found in reference 1. Since NASA-IGES is a subset of IGES, CAD systems have no problems in accepting files in NASA-IGES format. The NASA-IGES standard also specifies an even simpler subset called NASA-IGES-Nurbs-Only format which represents geometries solely with NURBS and trimmed NURBS. This single representation of geometry relieves CFD processes from complicated geometry issues, and yet, grants CFD processes the access of high quality geometry data.

Even though CAD systems can read in NASA-IGES files, they do produce files containing entities beyond NASA-IGES. To reduce the burden on grid generators, a second step was taken by the Subcommittee: a translator that converts a general IGES file to NINO format was developed. Along with the translator, a geometry viewer was also written to allow inspection and manipulation of NINO files. The translator and viewer are detailed in the following sections.



PRECEDING PAGE BLANK NOT FILMED

## NASA-IGES TRANSLATOR

The translator reads in most geometry entities from an IGES file (Table I). It performs conversion on those entities that require conversion. The conversion map is in Table II.

### Translator Behavior Control

An IGES entity (the child) can be "physically depedent" on another entity (the parent). The child entity can exist only when the parent entity exists. An example is that each constituent curve is physically dependent on a Composite Curve entity. The final model space location of a physically dependent entity depends on the transformation matrix of its parent (and grandparents, etc., if any). Normally, the translator discards any entity whose parent is not read in (not in Table I) or whose parent is discarded, even if the entity itself is in Table I. This behavior conforms to the IGES specification. However, an option is provided in the translator to override this behavior. When the option is used, the translator will convert a physically dependent entity without parent. The translator will also change such entities to be physically independent. However, the model space location of the entity in the output may not be correct, since the transformation matrix of its parent (grandparents) may not be available.

Normally the output of the translator contains the following entities: 124, 126, 128, 141, 142, 143, 212 (Form 0), 402 (Forms 1, 7, 14, 15), 406 (Form 15). A Subfigure Instance (Entity 408) is instantiated and converted to a group entity (Entity 402) containing a copy of all the geometries (converted) in the subfigure. This behavior eliminates instances from the output file and conforms to NINO format. The advantage of this format is that CFD processes which handle the NINO files do not have to deal with instances, which can be quite complex. The disadvantage of instantiating instances is that the output file may become quite large. The current translator contains an option that allows the user to suppress the instantiation of instances. When this option is used, the Subfigure Definition (Entity 308) and Subfigure Instance (Entity 408) are written to the output file.

### Conversion Approximation

Most of the conversions listed in Table II are mathematically exact (ref. 4). When exact conversions do not exist, approximations are used. Tolerance control is provided to the user through a tolerance variable in the translator resource file. Three kinds of approximation may take place. The first kind is for offset curves and offset surfaces. In general offset curves and surfaces can not be represented in the forms of its progenitors (ref. 5). For example, the offsets of bspline curves/surfaces are generally not bspline curves/surfaces. To convert offset entities to bsplines, approximation to the offset is required.

The second kind of approximation is for ruled surfaces when the rail curves of the ruled surfaces can not be converted to bsplines parametrically identically (ref. 6). An example is that of a ruled surface constructed between a circular arc and a line. Since the parametrization of arcs in IGES is defined trigonometrically, it is not rational. Hence, it is not possible to represent the parametrization with bsplines. IGES also allows ruled surfaces be defined by arc length parametrizations of the rail curves. It is well known that nearly all the arc length parametrizations are not rational.

The third kind of approximation occurs on trimmed or bounded surfaces when the parametrization of the underlying surfaces cannot be converted to bsplines exactly. This may happen when the underlying surfaces are ruled surfaces, surfaces of revolution, tabulated cylinders, and offset surfaces. When the parametrization of the surfaces cannot be converted exactly to bsplines, the parameter space curves of the trimmed/bounded surfaces need to be regenerated. Approximations are used for this regeneration.

# NASA-IGES VIEWER

## Overview

NIGESview is an application that permits the viewing and manipulation of objects read from a NINO file. The program reads in the NASA-IGES-Nurbs-Only subset of IGES and converts the resulting geometry into SGI's Open Inventor format which is then displayed. The input file should conform to the IGES Version 5.2 Specification (ref. 3). It is intended to be used in conjunction with the NIGEStranslator (Figure 1). NIGESview2.0 was developed using Open Inventor-2.0 and compiled on an SGI Indigo$^2$/ Extreme running IRIX-5.2 with 64Mbytes of RAM and 132Mbytes of swap space.

Open Inventor is a C++ API that rests on top of OpenGL and implements a display-list oriented graphical structure of your scene. This is stored as a "scene graph" and contains all of the items relevant to the geometry as well as the geometry itself. Many of the tedious aspects of OpenGL are removed so that the developer may concentrate on the application rather than the idiosyncrasies and event-handling nature of a low-level graphics API. The Open Inventor libraries should be available on vendor platforms other than SGI some time in 1995.

One of the goals of NIGESview is to eliminate the need for an expensive CAD package just to verify the correctness of NINO files. The other is to provide very simple editing of files to delete entities or merge entities from multiple files. In addition, few CAD packages have the ability to view entities 143 and 141 (Bounded surfaces), which NIGESview can display and manipulate.

The graphical interface permits a variety of editing features to manipulate the geometry in NINO files. The modified geometry can then subsequently be written out to either Inventor or NINO files. For example, if an IGES file contains a wing-body aircraft configuration, but also contains engine struts and cowls, these can be selected and deleted. Then the remaining geometry may be written to another NINO file.

During parsing of NINO files, entities in the input file that are not part of the NINO specification are thrown away. Entities that are not used for geometry viewing (e.g., properties, groups, etc.) will not be displayed, but are kept in memory so that they may be written to a NINO file, if the user chooses so. Physically dependent entities are ignored until the parent of the entity is parsed, then it is processed. If the entity has no parent, then the entity is lost, which is acceptable since the file is actually invalid. Note that the NIGEStranslator would not write out such an invalid file.

As each entity is parsed, it is converted to a group of Open Inventor "nodes," which represent graphical/attribute objects that we want to place in our scene graph. NIGESview uses the Open Inventor concept of a "kit" to represent an IGES geometry. The kit contains Open Inventor equivalents of IGES concepts such as subscript names (for info only), line widths and fonts, color, transforms, and of course, control points and NURBS data. So a kit is an entire encapsulation of an IGES geometry and its directory section information (Figure 2). See references 1 and 3 for details of IGES geometry attributes.

Two problems were encountered when we used Inventor for NURBS display. First, Inventor (also OpenGL) is extremely slow when the NURBS surface(s) is large. Second, Inventor can not render bsplines of order greater than 8. We circumvented these problems partially by tessellating unbounded NURBS surfaces ourselves. When unbounded NURBS surfaces are encountered, NIGESview calls routines to evaluate and tessellate the surface into a quadrilateral mesh. The resolution of the evaluation is determined using heuristics at parsing time, but the user may change this interactively after the file is loaded - to increase or decrease resolution. Due to the excellent performance of our tessellation routines and the high rendering

speed of quadrilaterals by Inventor, NIGESview can handle large unbounded NURBS surfaces reasonably well.

NIGESview supports two kinds of "resolution" or complexity of geometries. The one for unbounded surfaces (discussed above) is called "Unbounded Complexity". Everything else in the scene (curves, bounded surfaces, text) is driven by "General Complexity," which is controlled by one value. A dialog box is provided for users to change either or both complexities at any time. In addition, a popup menu in the drawing area provides even more resolution flexibility with drawing styles such as Bounding Box, Low-res, Points, and Wireframe.

## Editing

After a file is loaded and converted to Open Inventor, there exists two copies of the data in memory: one for Inventor, the other for IGES. When users modify geometries with NIGESview editors, both the Inventor and IGES data are modified. For example, when a user deletes an entity, not only is the Inventor data deleted, but also the IGES data in the model; if the NINO file is written out, the entity will not be in the output. For cases where the user translates, rotates, or scales the geometry, the result of writing an IGES file is that an IGES Transformation Matrix entity (124) is created or added to the modified entity. All IGES protocols regarding order of matrix multiplication, and so forth, are obeyed (Figure 3).

Editing Physically dependent entities (bounded surfaces) requires more caution. There are many IGES entities associated with one bounded surface - the trimming curves (parameter and model space NURBS curves), the coordinates, and the underlying surface. When copying/pasting/deleting a surface, NIGESview provides an interface dialog to specify how to process physically dependent surfaces and their "relatives".

Also available to the user is the ability to inquire IGES information that comes from the input file. The Start Section, for example, can be edited like standard text, and it will be reformatted and written out to the output IGES file when a user requests it. Data such as the list of read entities and the global section can not be directly modified, but can be viewed in a separate dialog window.

## Issues

One of the difficulties of any package is dealing with large files having many large entities (or worse - lots of trimmed surfaces). NIGESview is no exception, but if the file can be parsed by lower level libraries, then the user can improve the usability of NIGESview with the complexity options discussed earlier. NIGESview can be sensitive to large files because there are always two copies in memory at any time. IGES data is not structured in a way that allows OpenGL or Open Inventor to merely reference the data - they must copy the data.

There are features provided by Open Inventor which are virtually free of (engineering) cost. These features are included in NIGESview for high quality image rendering. A brief list includes the following: editing of 6 axial lights, surface material editing (color, transparency, lighting reflective properties), background color, depth cueing, and choice of box or line highlighting of picked objects. These types of attributes are NOT saved to NINO files (IGES provides no support for these).

NIGESview limitations include features such as transforming groups of entities at once, editing raw control points of NURBS curves and surfaces, automated evaluation/tessellation of bounded surfaces and curves, saving color modifications of entities, and UNDO operations while editing.

## DISTRIBUTION

The NIGES software including NIGEStranslator, NIGESviewer, and a set of IGES test files is available from NASA Ames Research Center. The software runs on SGI IRIX 5.2 machines. All software distribution includes source code. In case recompilation is necessary, SGI C++3.1 is required; in addition, OpenInventor 2.0 development environment is required for NIGESViewer. In the near future the software will be forwarded to COSMIC for distribution.

## ACKNOWLEDGEMENT

The authors would like to thank NASA for the support of this project. Special thanks are attributed to Matthew Blake, Dr. Bill Van Dalsem, Francis Enomoto, and Cathy Maksymiuk at NASA Ames Research Center.

## REFERENCES

1. M. Blake, et al, NASA Geometry Data Exchange Specification for Computational Fluid Dynamics (NASA IGES), NASA RP 1338, April 1994.

2. M. Blake, et al, The NASA-IGES Geometry Data Exchange Standard, *NASA Workshop on Software Systems for Surface Modeling and Grid Generation*, NASA CP 3143, April 1992.

3. Initial Graphics Exchange Specification (IGES) Version 5.2, US PRO/IPO-100.

4. L. Piegl, W. Tiller, Curve and Surface Constructions Using Rational B-Splines, *Computer-Aided Design*, Vol 19, No 9, 1987, pp 485-498.

5. B. Pham, Offset Curves and Surfaces:A Brief Survey, *Computer Aided Design*, Vol 24, No 4, 1992, pp 269-279.

6. J. Chou, Higher Order Bezier Circles, *Computer-Aided Design*, Vol 27, No 3, 1995, pp 250-259.

**Table I**: Entities in the IGES input file read in by NIGEStranslator.

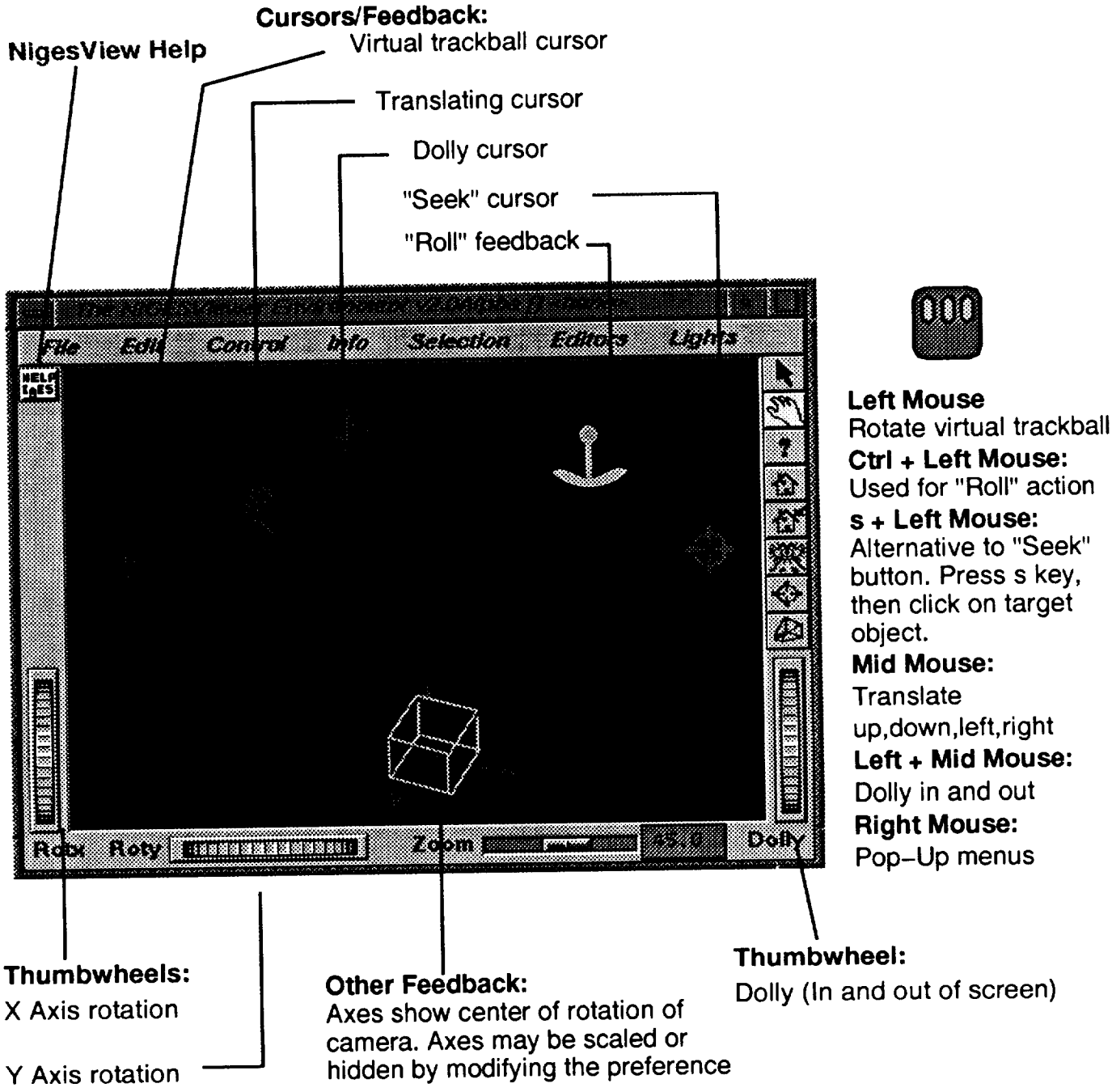
| IGES Entity # | Entity Name |
|---|---|
| Entity 100 | Circular Arc |
| Entity 102 | Composite Curve |
| Entity 104 | Conic Arc |
| Entity 106 (Forms 1,2,3,11,12,13,63) | Copious Data |
| Entity 108 (Form 1) | Bounded Plane |
| Entity 110 | Line |
| Entity 112 | Parametric Spline Curve |
| Entity 114 | Parametric Spline Surface |
| Entity 116 | Point |
| Entity 118 | Ruled Surface |
| Entity 120 | Surface of Revolution |
| Entity 124 | Transformation Matrix |
| Entity 126 | Rational B-Spline Curve |
| Entity 128 | Rational B-Spline Surface |
| Entity 130 | Offset Curve |
| Entity 140 | Offset Surface |
| Entity 141 | Boundary |
| Entity 142 | Curve on Parametric Surface |
| Entity 143 | Bounded Surface |
| Entity 144 | Trimmed Parametric Surface |
| Entity 212 (Form 0) | General Note |
| Entity 314 | Color Definition |
| Entity 402 (Forms 1, 7, 14, 15) | Associativity Instance, |
| Entity 406 (Form 1) | Property, Definition Levels |
| Entity 406 (Form 15) | Property, Name |
| Entity 308 | Subfigure Definition |
| Entity 408 | Singular Subfigure Instance |

Table II: NIGEStranslator Conversion Map

| From Entity Name | Type (,Form) | To Entity Name | Type (,Form) |
|---|---|---|---|
| Circular Arc | Type 100 | Rational B-Spline Curve | Type 126, Form 2, Degree 2, PROP1 1, PROP3 0, PROP4 0 |
| Composite Curve | Type 102 | Rational B-Spline Curve | Type 126, Form 0 |
| Conic Arc | Type 104 | Rational B-Spline Curve | Type 126, Form 3, 4, or 5 as appropriate, Degree 2, PROP1 1, PROP3 1 for parabola, PROP3 0 for others, PROP4 0 |
| Copious Data | Type 106, Form 1 or 11 | Rational B-Spline Curve | Type 126, Form 0, Degree 1, PROP1 1, PROP3 1, PROP4 0 |
| Copious Data | Type 106, Form 2, 3, 12 or 13 | Rational B-Spline Curve | Type 126, Form 0, Degree 1, PROP3 1, PROP4 0; Note: The information about the vectors associated with the points will be lost. |
| Copious Data | Type 106, Form 63 | Rational B-Spline Curve | Type 126, Form 0, Degree 1,PROP1 1, PROP2 1, PROP3 1, PROP4 0 |
| Plane | Type 108, Form 1 | Bounded Surface/ Boundary | Type 143, Form 0/ Type 141, Form 0 |
| Line | Type 110 | Rational B-Spline Curve | Type 126, Form 1, Degree 1,PROP1 1, PROP3 1, PROP4 0 |
| Parametric Spline Curve | Type 112 | Rational B-Spline Curve | Type 126, Form 0 |
| Parametric Spline Surface | Type 114 | Rational B-Spline Surface | Type 128, Form 0 |
| Ruled Surface | Type 118 | Rational B-Spline Surface | Type 128, Form 8 |
| Surface of Revolution | Type 120 | Rational B-Spline Surface | Type 128, Form 6 |

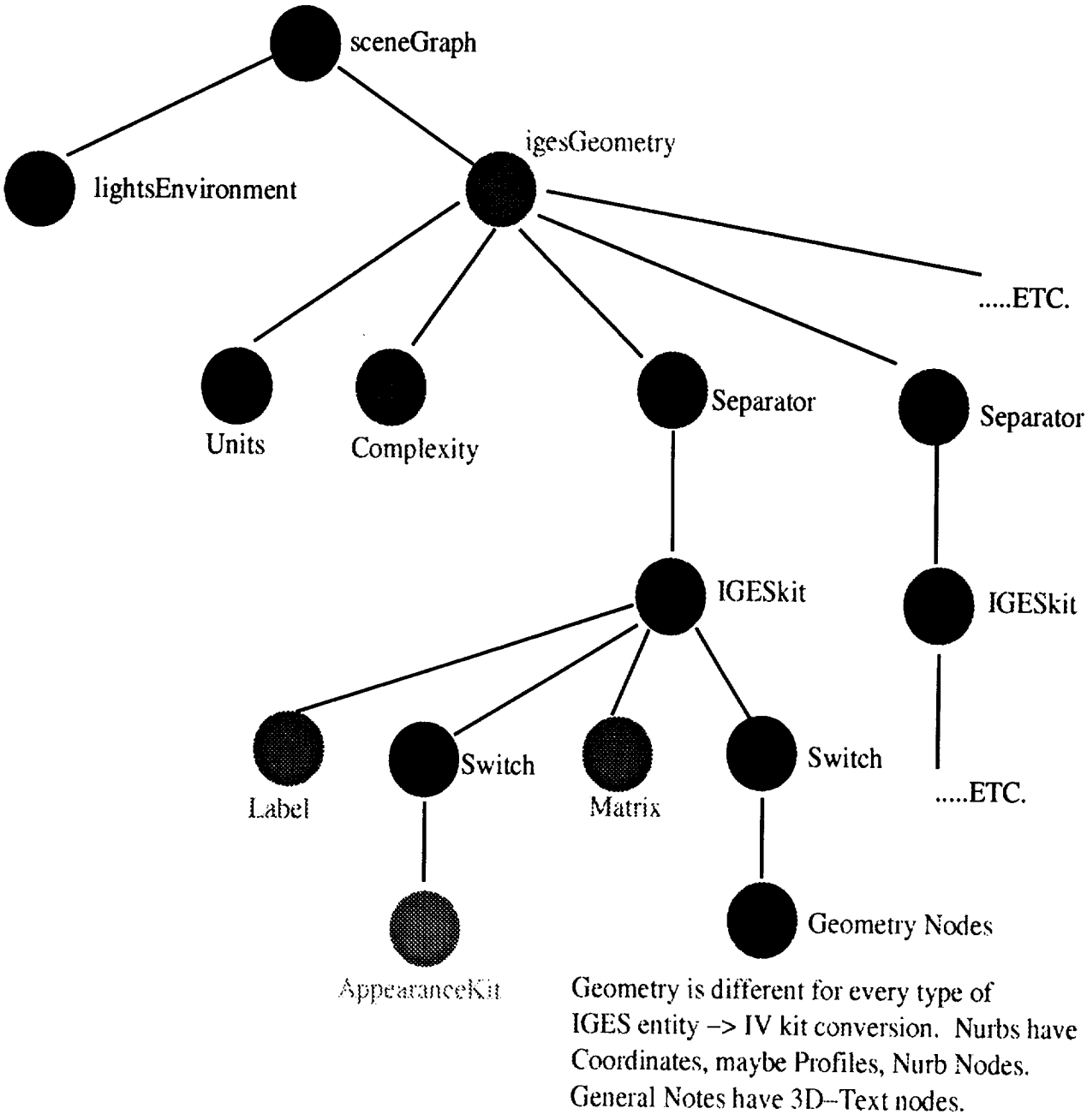
Table II: NIGEStranslator Conversion Map

| From Entity Name | Type (,Form) | To Entity Name | Type (,Form) |
|---|---|---|---|
| Tabulated Cylinder | Type 122 | Rational B-Spline Surface | Type 128, Form 7 |
| Offset Curve | Type 130 | Rational B-Spline Curve | Type 126, Form 0 |
| Offset Surface | Type 140 | Rational B-Spline Surface | Type 128, Form 0 |
| Curve On Parametric Surface | Type 142 (Not part of 144) | Curve On Parametric Surface | Type 142, with all curves and surfaces converted to B-Splines |
| Curve On Parametric Surface | Type 142 (Part of 144) | Boundary | Type 141, with all curves and surfaces converted to B-Splines |
| Trimmed Surface | Type 144 | Bounded Surface | Type 143, with all curves and surfaces converted to B-Splines |
| Definition Levels | Type 406, Form 1 | The entity with this property is placed in the first level identified by this Definition Levels entity. | none |
| Singular Subfigure Instance | Type 408 | A group (Associativity Instance Entity) of the geometry using original entities. These entities are then converted as specified in these Conversion Maps. This conversion takes place if the "-s" flag is not used | Type 406 |




## Figure 1.

This viewer uses a virtual trackball to rotate the view. The point of rotation is by default the center of the scene bounding box, but can be placed anywhere in the scene. This viewer also allows you to translate in the screen plane, as well as dolly in and out (forward/backward movement).




## Figure 2:

Inventor uses a directed acyclic graph to represent data. To render a scene, Inventor traverses the graph depth-first, left to right. The graph is easy to manipulate and only needs created once, which for NIGESview, is when the IGES file is parsed.

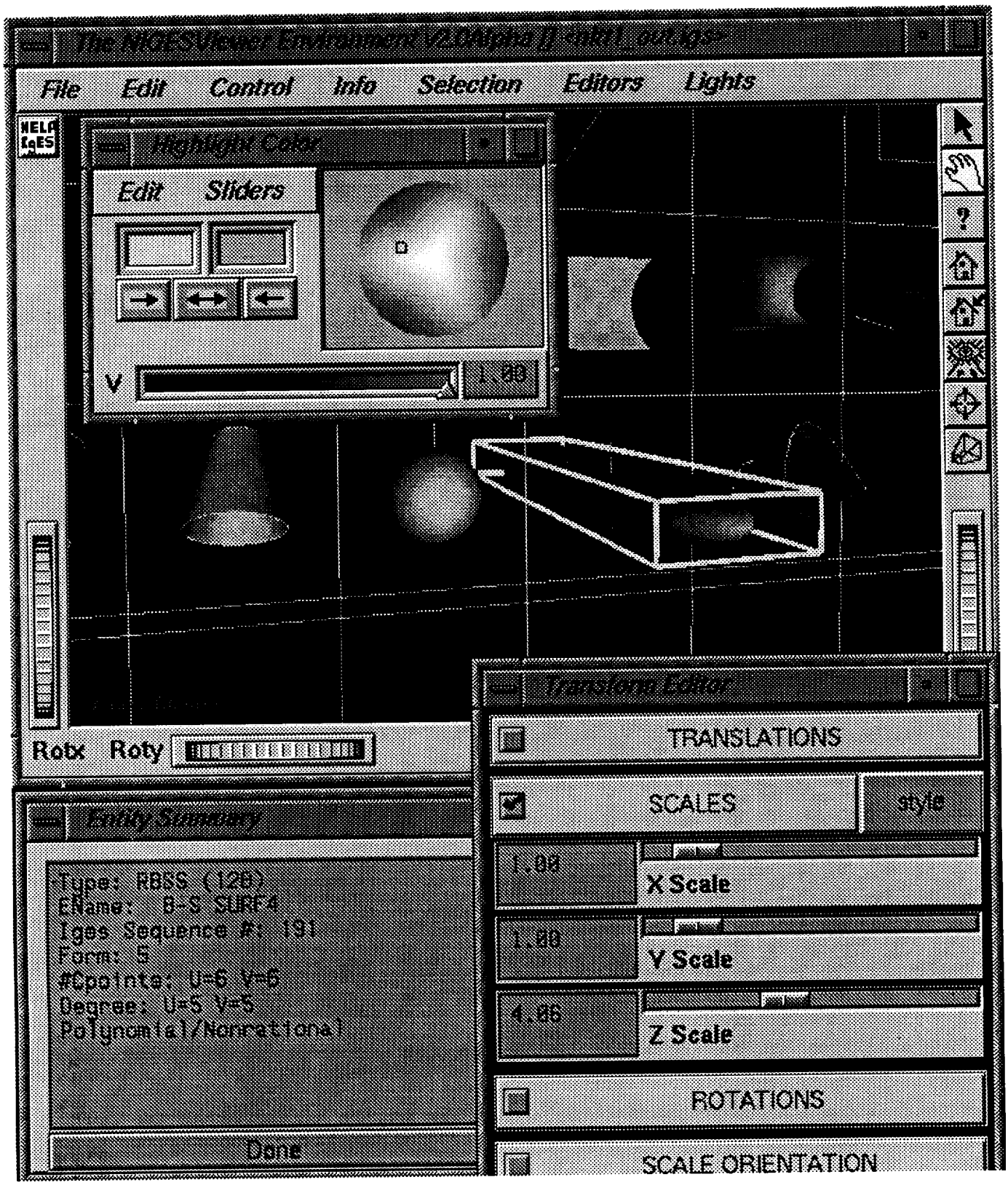



**Figure 3**: example of a surface being edited by NIGESview Inventor options. Note that the Scaling change **will** be saved to an IGES geometry file.

# REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.

| 1. AGENCY USE ONLY (Leave blank) | 2. REPORT DATE | 3. REPORT TYPE AND DATES COVERED |
|---|---|---|
| | March 1995 | Conference Publication |

**4. TITLE AND SUBTITLE**

Surface Modeling, Grid Generation, and Related Issues in Computational Fluid Dynamic (CFD) Solutions

**5. FUNDING NUMBERS**

WU-505-62-52

**6. AUTHOR(S)**

Yung K. Choo, compiler

**7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)**

National Aeronautics and Space Administration
Lewis Research Center
Cleveland, Ohio 44135-3191

**8. PERFORMING ORGANIZATION REPORT NUMBER**

E-9458

**9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)**

National Aeronautics and Space Administration
Washington, D.C. 20546-0001

**10. SPONSORING/MONITORING AGENCY REPORT NUMBER**

NASA CP-3291

**11. SUPPLEMENTARY NOTES**

Proceedings of the workshop on Surface Modeling, Grid Generation, and Related Issues in Computational Fluid Dynamic (CFD) Solutions held at NASA Lewis Research Center, May 9-11, 1995. Responsible person, Yung K. Choo, organization code 2610, (216) 433-5868.

**12a. DISTRIBUTION/AVAILABILITY STATEMENT**

Unclassified - Unlimited
Subject Category 02

This publication is available from the NASA Center for Aerospace Information, (301) 621-0390.

**12b. DISTRIBUTION CODE**

**13. ABSTRACT** *(Maximum 200 words)*

The NASA Steering Committee for Surface Modeling and Grid Generation (SMAGG) sponsored a workshop on surface modeling, grid generation, and related issues in Computational Fluid Dynamics (CFD) solutions at Lewis Research Center, Cleveland, Ohio, May 9-11, 1995. The workshop provided a forum to identify industry needs, strengths and weaknesses of the five grid technologies (patched structured, overset structured, Cartesian, unstructured, and hybrid), and to exchange thoughts about where each technology will be in 2 to 5 years. The workshop also provided opportunities for engineers and scientists to present new methods, approaches, and applications in SMAGG for CFD. This Conference Publication (CP) consists of papers on industry overview, NASA overview, five grid technologies, new methods/approaches/applications, and software systems.

| 14. SUBJECT TERMS | 15. NUMBER OF PAGES |
|---|---|
| Surface modeling; Grid generation; Computational fluid dynamics | 865 |
| | 16. PRICE CODE: A99 |

| 17. SECURITY CLASSIFICATION OF REPORT | 18. SECURITY CLASSIFICATION OF THIS PAGE | 19. SECURITY CLASSIFICATION OF ABSTRACT | 20. LIMITATION OF ABSTRACT |
|---|---|---|---|
| Unclassified | Unclassified | Unclassified | |